EXHIBIT 1

Filed Under Seal

DECLARATION OF SAMRAT BHATTACHARJEE, PH.D

I, Samrat Bhattacharjee, declare as follows:

1. I have been retained to testify as an expert in this action on behalf of Google, LLC ("Google"). I submit this Declaration in support of Google's Motion for Summary Judgment Regarding Claim 13 of U.S. Patent No. 9,967,615 ("Motion") against Sonos, Inc. ("Sonos"). I am being compensated for my work on this case at my standard consulting rate of $700/hr. My compensation is not contingent upon the results of my analysis or the substance of my testimony.

I. QUALIFICATIONS

2. I provide below an overview of my background and qualifications.

3. I earned a Ph.D. in Computer Science from Georgia Institute of Technology in 1999. Before that, I earned Bachelor of Science degrees in Mathematics and Computer Science from Georgia College and State University in 1994. I was a Teaching Assistant at the Georgia Institute of Technology from 1994 to 1995 and an Instructor in 1998. From 1999 through 2005, I was an Assistant Professor at the University of Maryland, College Park, in the Department of Computer Science, and then an Associate Professor with tenure from 2005 through 2009. In 2006, I was a Visiting Professor at Max Planck Institüt für Software Systems in Saarbrücken, Germany. In 2007, I was a Visiting Researcher at AT&T Labs in Florham Park, New Jersey. Since 2009, I have been a tenured Professor at the University of Maryland. My teaching and research have focused on networking, multimedia streaming, and other aspects of the technologies pertaining to the '615 patent. For instance, I have published multiple papers in the field of video streaming over the Internet. While at AT&T Research, I worked on a scalable system for video distribution, which resulted in publications, and a patent (US8752100B2).

4. I served as a reviewer for ACM/IEEE Transactions on Networking, IEEE Journal on Selected Areas in Communications, Computer Communications Journal (Special Issue on Network Security), ACM Transactions on Computer Systems, Performance Evaluation Journal, Computer Communications Review, European Transactions on Telecommunications, IEEE Transactions on Parallel and Distributed Systems, and ACM Transactions on Internet Technology.

5. I am the author of numerous publications in the field of networking, including journal articles, book chapters, publications in proceedings, technical reports, and invited papers.

6. I have been active in a number of professional organizations and conferences. I have served with the National Science Foundation (NSF) Workshop on Network Testbeds, the NSF Networking Research Panel, the Department of Education High Performance Networking Panel, and as an Evaluator for the Intel Science Talent Search.

7. I have received several honors and awards. These include: the Alfred P. Sloan Jr. Fellowship; the Best Paper Award, 14th Annual IEEE International Conference on High Performance Computing (HiPC) (with Vijay Gopalakrishnan, Ruggero Morselli, Peter Keleher, and Aravind Srinivasan); the Best Paper Award, 7th IEEE/ACM Conference on Grid Computing (with Jiksoo Kim, Byomsuk Nam, Peter Keleher, Michael Marsh, and Alan Sussman); and the NSF CAREER Award. I also received Teaching Excellence Awards in 2004, 2008, and 2012.

8. In 2017, along with colleagues from Cornell University and the Max Planck Society in Germany, I started an annual school for introducing undergraduate students from across the world to research. This school is partially supported by the NSF and had student participants from 29 countries. I am also the co-director of a new joint Ph.D program in computer science between the University of Maryland and the Max Planck Society in Germany.

II. LEGAL STANDARDS

9. I have been informed by counsel of various legal standards. I set forth my understanding below.

A. Person of Ordinary Skill in the Art

10. I have been advised that patent claims are reviewed from the point of view of a hypothetical person of ordinary skill in the art ("POSITA") at the time of the filing of the patent.

B. Non-Infringement

11. It is my understanding that an accused system and/or method literally infringes a claim of a patent if and only if the accused device or system contains every element of the claim. If the accused device or system does not contain one or more elements or steps recited in the claim,

then there is no literal infringement of that claim.

12. I further understand that determining whether there is infringement of a patent includes two steps. First, each asserted claim must be construed to determine its proper scope and meaning to a POSITA. Second, I understand that once the scope of the asserted claims has been determined, the construed claims are compared with the accused product or service to determine whether every limitation of the claims is found. Unless every limitation is present in the accused product or process, there is no infringement.

13. I have been informed that if a given claim limitation is not literally present in an accused system or method, the accused feature or step in the system or method may nevertheless meet this limitation under the doctrine of equivalents. I have been informed that the test for infringement under the doctrine of equivalents is whether the accused system or method possesses structure or steps that a POSITA would think, at the time of infringement, is insubstantially different from the claim limitation.

14. I understand that whether the difference is insubstantial can be analyzed using the "function–way–result" test, wherein the accused system is deemed to be equivalent to a claim limitation if the accused feature performs substantially the same function in substantially the same way in order to achieve substantially the same result. In deciding whether any difference between a claim requirement and the accused feature is insubstantial, I understand that I can consider whether, at the time of the alleged infringement, POSITAs would have known of the interchangeability of the accused feature or step with the claimed element. However, the known interchangeability between the claimed element and the feature or step of the accused system or method is not necessary to find infringement under the doctrine of equivalents.

C. Legal Standard for Prior Art

15. I understand that a granted patent is presumed to be valid. I understand that the presumption of validity can be overcome if clear and convincing evidence is presented that proves the patent is invalid. I further understand that a patent or other publication must first qualify as prior art before it can be found to invalidate a patent claim. I understand that a U.S. patent qualifies

as prior art to the asserted patent if the application for that patent was filed in the United States before the alleged invention of the asserted patent. I understand that to qualify as prior art, a reference must contain an enabling disclosure that allows one of ordinary skill to practice the claims without undue experimentation. I understand that a system or device qualifies as prior art to the asserted patent if it was in "public use," meaning it was accessible to the public or was commercially exploited, before the alleged invention of the asserted patent.

D. Anticipation

16. I understand that, once the claims of a patent have been properly construed, determining anticipation of a patent claim requires a comparison of the properly construed claim language to the prior art on a limitation-by-limitation basis. I understand that a prior art reference "anticipates" an asserted claim, and thus renders the claim invalid, if all elements of the claim are disclosed in the reference, either explicitly or inherently (*i.e.*, necessarily present or implied).

E. Obviousness

17. I understand even if a claim is not anticipated, it is still invalid if the differences between the claimed subject matter and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a POSITA.

18. I understand that an obviousness determination includes the consideration of factors such as: (1) scope and content of the prior art, (2) differences between the prior art and the asserted claims, (3) level of ordinary skill in the pertinent art, and (4) existence of secondary considerations or objective indicia of obviousness or non-obviousness. I understand that an obviousness determination can be based on a single prior art reference or a combination of prior art references. I understand that the prior art may provide a suggestion, motivation, or reason to combine or modify the teachings of the prior art, or that such a reason may come from other sources (*e.g.*, knowledge of a POSITA, common sense, and market forces). I understand that an obviousness determination requires a reasonable expectation of success in achieving the claimed invention.

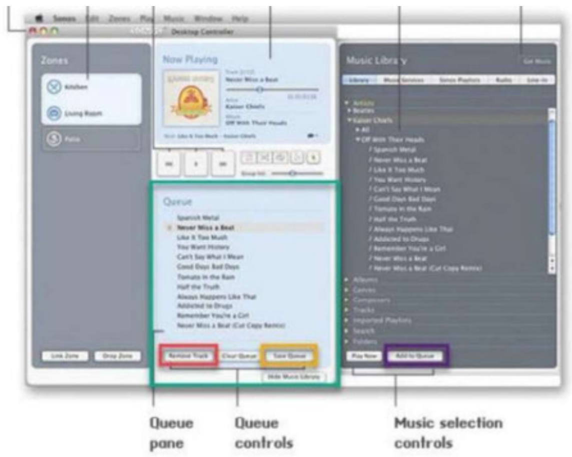
III. BRIEF OVERVIEW OF THE '615 PATENT

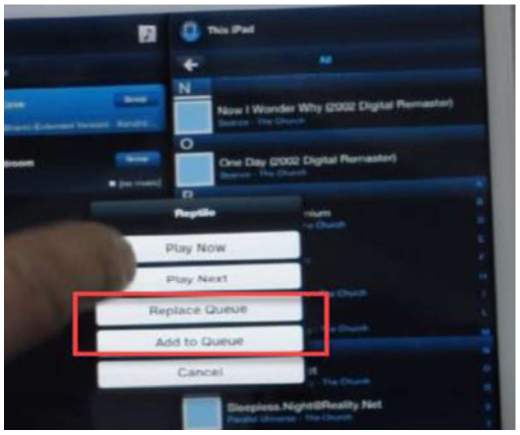
19. The '615 patent was filed on February 23, 2015, as a continuation of an application

filed December 30, 2011. I understand that Sonos has alleged that Claim 13 has a July 15, 2011 invention date. The patent discloses a playback system comprising a computing device or controller and playback device. Figure 1 illustrates such a playback system in a home, with a controller 130 and playback devices 102-124. *Id.*, 3:17-36, Fig. 1.

20. The patent further discloses that the controller of the system provides a graphical user interface (GUI) for "navigat[ing] a playlist of many multimedia items and [] control[ing] operations of one or more [playback devices]." '615 Pat., 3:30-33, 9:10-48.

21. The patent teaches that the controller may be the "Sonos Controller for iPhone," the "Sonos Controller for iPad," the "Sonos Controller for Mac or PC." *Id.*, 4:66-5:11. Using the controller, users can "queue up music" and edit and manage that queue. '615 Pat., 17:14-15, 16:25-31, 13:22-28. The Sonos Controller manual from 2011 define a "music queue" as follows: "[w]hen you make music selection they are added to a list of tracks called a music queue." Dkt. No. 200-12 at 5-1. The image on the left below shows the "queue" in the Sonos Controller for Mac or PC, which includes the ability for users to "add" (purple box) "remove" (red box) and "save" (yellow box) a queue that contains an ordered list of tracks for playback. Kyriakakis Decl., ¶22. The image on the right shows the "queue" in Sonos's iPad Controller application, which also allowed users to add and edit a queue containing a list of tracks.





Dkt. No. 200-12 (Sonos Controller Guide) at 5-1 (left image); https://www.youtube.com/watch?v=o586iKAZAf8 (uploaded on 2/6/2013) (right image)

22. The specification also teaches that users may switch playback of content from the controller to a playback device in the local network. ’615 Pat., 1:11-29, 1:66-20, 12:44-67, 13:14-22, 13:54-56. The playback device can be configured to playback multimedia items by fetching the content using “a uniform resource locator (URL) that specifies an address to a particular audio track in the cloud.” *Id*., 11:60-12:4, 12:53-55, 13:13-22, 13:54-59.

IV. STATE OF THE ART

23. In this section, I provide an overview of the state of the art at the time of Sonos’s alleged invention date (July 15, 2011).

24. By July of 2011, the ability to play back audio and/or video content that was stored on servers in the cloud was well-known. Numerous cloud-based streaming services were available. For instance, YouTube launched in 2005 and permitted users to playback video content that was stored in the cloud. Netflix launched its streaming video service by 2007, which provided an online movie service. Various online music and radio streaming services were also available, including last.fm, Spotify, Pandora and Rhapsody. These streaming services allowed users to playback online content using their mobile device or computer. The ’615 patent itself concedes that playback of cloud-based content was known: “a user can access audio, video, or both audio and video content over the Internet through an online store, an Internet radio station, an online music service, an online movie service, and the like, in addition to the more traditional avenues of accessing audio and video content.” ’615 patent, 1:32-35.

25. The ability to transfer playback from a mobile device to a playback device (such as a TV or speaker)—including the ability to select individual playback devices for transfer from the user interface of a mobile phone or tablet—was also a known technique, as I discuss below.

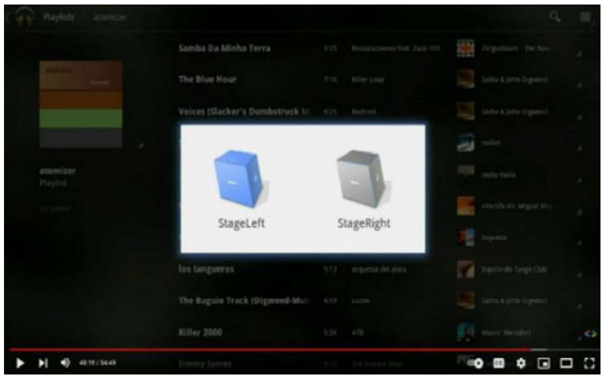
1. Google’s Project Tungsten And YouTube Remote Prior Art

26. Google’s own products implemented playback transfer from a mobile device to a playback device, including the ability to select a particular playback device for transfer.

27. For example, Google’s Project Tungsten included a tablet that allowed a user to transfer playback to a Tungsten device that was connected to speakers for playback. Project

Tungsten was demonstrated at the May 10, 2011 Google I/O,[1] shown in this video: https://www.youtube.com/watch?v=OxzucwjFEEs&t=2808s at 46:47-48:50.

28. The images below are taken from a video uploaded to YouTube on May 10, 2011 of the I/O and show the Tungsten devices (left image), as well as the tablet that could be used to select a particular Tungsten device (StageLeft or StageRight devices) for transferred playback:



https://www.youtube.com/watch?v=OxzucwjFEEs&t=2808s at 47:01 (left) and 48:19 (right). As Google engineer Joe Britt stated at the I/O, "the tablet can direct music to one or more Tungsten boxes like the ones we have here." https://www.youtube.com/watch?v=OxzucwjFEEs&t=2808s at 48:18-48:25. "When Anand tapped on those buttons [on the tablet], the music stream was sent transparently from one box to another. Since the boxes are running Android, they just pull the music directly from the music library in the cloud." https://www.youtube.com/watch?v=OxzucwjFEEs&t=2808s at 48:36-48:50.

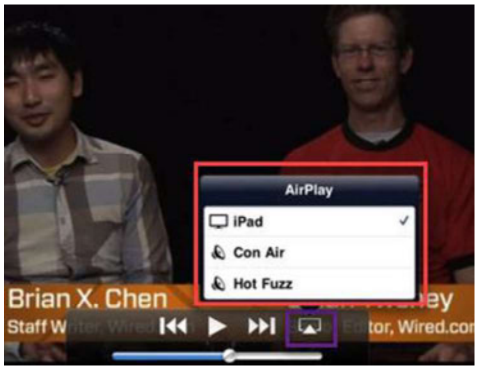
29. Similarly, Version 1.0 of the YouTube Remote application was released on November 9, 2010. Bobohalma Decl., ¶3. As I discuss later in this report (Section VII), the YouTube Remote permitted users to transfer playback from their mobile device (*e.g.*, an Android phone) to one or more playback devices (television screens). And a patent obtained on the YouTube Remote—U.S. Patent No. 9,490,998—also discloses that a user may select a particular playback device ("controlled devices") from the display of a mobile device ("the remote control"). '998 patent at 10:62-11:6 ("In some examples, the user may also utilize the remote control

[1] Google I/O is an annual Google developer conference typically held in May.

application of remote control 75 to select one or more previously paired controlled devices").

2. Prior Art Devices Enabled With Apple AirPlay

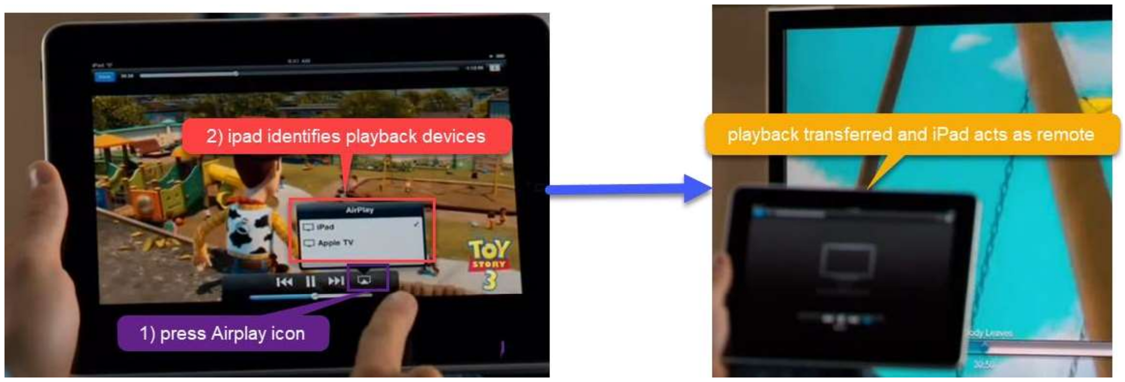
30. As another example, Apple Airplay launched in 2010 as part of Apple's iOS 4.2 release. *See* https://www.apple.com/newsroom/2010/11/22Apples-iOS-4-2-Available-Today-for-iPad-iPhone-iPod-touch/. Airplay functionality was built into Apple's iPhone and iPad devices and allowed a user to transfer playback of music from a phone to television or speaker. The user would select a selectable "AirPlay" icon and then further select an individual playback device from a list of available devices shown on the screen to transfer playback. For instance, the image above shows an iPad with the Airplay icon (purple box) selected and available devices, for example "Con Air" and "Hot Fuzz" (red box) available for selection. Ex. 12 [Hidden Secrets of Airplay, 11/23/2010] (image annotated above to add red box).



31. This operation of Airplay is shown in various YouTube videos, for example:

- **Airplay Video #1** (https://www.youtube.com/watch?v=fGMdg13YWB8): Apple iOS 4.2 Airplay (uploaded, November 22, 2010)
- **Airplay Video #2** (https://www.youtube.com/watch?v=orfD45VQB1c): Apple Airplay Demo for the iPad, iPhone and iPod Touch (uploaded Nov. 24, 2010)

32. For instance, the images below are from **Airplay Video #1**, uploaded to YouTube on November 22, 2010, and show a user (1) watching Toy Story 3 on their iPad, (2) tapping the "Airplay" icon (purple box annotation) to display available playback devices (red box annotation), for instance the "Apple TV," (3) selecting the Apple TV to transfer playback to the device, and (4) watching Toy Store 3 on the TV, while using his iPad to control the video on the TV:


2) ipad identifies playback devices
AirPlay
iPad
Apple TV
TOY STORY 3
1) press Airplay icon
playback transferred and iPad acts as remote

Airplay Video #1 at 0:35-0:50 (annotated to add callouts and purple and red boxes).

3. Prior Art Patent Publications

33. The ability to transfer playback to a particular playback device was also described in prior art patent publications. For example, U.S. Publication No. 2011/0131520 to Al-Shaykh (Al-Shaykh) was filed on November 29, 2010 and published on June 2, 2011. Al-Shayk also discloses a user interface 31 of a media application with a set of controls and indications 35 (Fig. 2) that can be used "for media transferring and sharing." The set of control and indications on the controller device may include a media transfer control 51 and a rendered selection control/indication 53, which a user may use to send playback to a selected location, such as the "living room" or "bedroom" selection shown in Figures 3 and 4, respectively. Al-Shayk, ¶¶ 99-100 ("The media transfer control 51 may be used to enable and/or disable the transfer of media content 15 from the media application to the target rendering device in the home network.").

FIG. 2

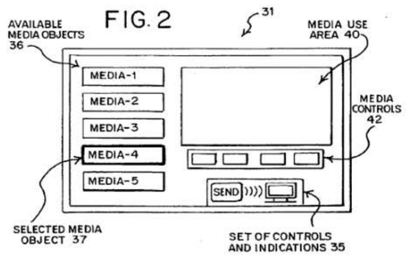


FIG. 3

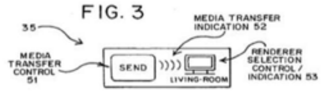
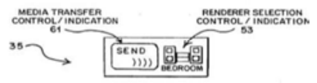
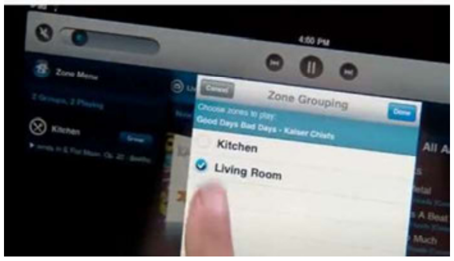

35
MEDIA TRANSFER INDICATION 52
MEDIA TRANSFER CONTROL 51
SEND
LIVING-ROOM
RENDERER SELECTION CONTROL / INDICATION 53

FIG. 4


MEDIA TRANSFER CONTROL / INDICATION 61
RENDERER SELECTION CONTROL / INDICATION 53
35
SEND
BEDROOM

4. Sonos's Own Prior Art

34. Sonos's own prior art products also included a controller (*e.g.*, an iPhone or tablet application) that would identify available devices for playback and allow a user to select which devices he or she wanted to playback music on. For instance, the '615 patent refers to various prior art Sonos controller applications, such



as the "Sonos Controller for iPad," that supported this feature. '615 patent at 4:52-5:11. On the right is a screen shot from a video uploaded to YouTube on June 22, 2010 that shows a Sonos Controller for iPad that has identified available playback devices ("Kitchen" and "Living Room" devices) and provides a display for the user to select a particular playback device. https://www.youtube.com/watch?v=aCWcl_f_uS0 at 0:55-1:09.

B. Person of Ordinary Skill in the Art

35. I understand that as part of the claim construction stage Google's and Sonos's experts submitted competing standards of a POSITA (Kyriakakis Decl., ¶ 37; Schmidt Decl., ¶ 29). I was a POSITA at the time of the alleged invention under both standards. The difference between these two definitions does not change my opinions. Further, I note that my opinions would not change if a slightly higher or lower degree of expertise were applied.

V. CLAIM CONSTRUCTION

36. I understand that Sonos's lawsuit against Google was initially filed in the Western District of Texas, and then transferred to this Court in California. I further understand that prior to transfer, the Texas court construed the following terms that appear in Claim 13:

Claim Term	Court's Construction (WDTx)
"multimedia" ['615 Patents]	Plain and ordinary meaning.
"network interface" ['615, '885 Patents]	Plain and ordinary meaning
"playback device" / "zone player" ['615, '033 Patents] / ['966 and '885 Patents]	Plain and ordinary meaning "Zone player" means "playback device"
"local area network" ['615 Patent]	Plain and ordinary meaning
"cloud" ['615 and '033 Patents]	Plain and ordinary meaning

37. I also understand that the parties have proposed competing constructions for certain additional terms that appear in Claim 13:

Claim Term	Google Proposed Construction	Sonos Proposed Construction
"playback queue"	"an ordered list of multimedia items that is selected by the user for playback"	Plain and ordinary meaning; no construction necessary
"resource locators"	"address of a resource on the Internet"	Plain and ordinary meaning; no construction necessary

38. I have applied Google's proposed construction and the Court's constructions in my analysis below. For terms that are not specifically addressed by the parties or Court, I have applied the plain and ordinary meaning to a POSITA. I have also addressed certain positions Sonos has taken for purposes of infringement, which appear to be inconsistent with the plain meaning.

VI. NON-INFRINGEMENT ANALYSIS

39. Claim 13 of the '615 patent is directed at a tangible, non-transitory computer readable storage medium that cause a control device to implement certain instructions. Sonos has accused over 1,500 different smartphones, tablets, and computers of being an infringing "control device." *See* Sonos January 20, 2022 Infringement Contentions, Appendix A. I understand Sonos to be alleging that the devices infringe when used with (1) Google Play Music; or (2) one of the following YouTube applications: YouTube, YouTube Kids, YouTube TV, and YouTube Music.

40. Based on my review and analysis of the facts in this case, as considered in light of my experience and expertise in the field, it is my opinion that Google does not infringe Claim 13 of the '615 patent. Further facts and analysis supporting my opinion are set forth below.

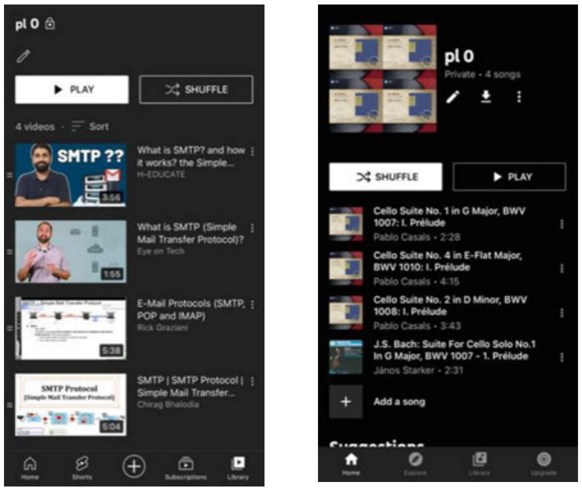
A. The Accused YouTube Applications Do Not Infringe Claim 13

1. Overview Of The Accused YouTube Applications

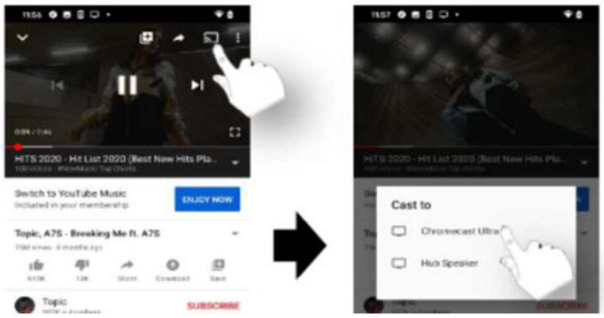
41. YouTube is a streaming video website. One way to watch YouTube is by downloading and installing a YouTube application on a mobile device that supports the application. The main application for YouTube is referred to as "YouTube Main" or just "YouTube," while YouTube also offers a separate, kid-friendly application called "YouTube Kids." In addition to online video streaming, YouTube provides an Internet television service known as "YouTube TV" and a music service called "YouTube Music." These services may also be accessed using a web browser on a computer or mobile device, or by downloading the "YouTube TV" and "YouTube Music" applications. While each of the YouTube, YouTube Kids, YouTube TV and YouTube Music application are separate application, I will collectively refer to

them as the accused "YouTube applications" in this declaration except where specifically noted.[2]

42. The YouTube and YouTube Music applications permit users to select media items (*e.g.*, videos or songs) and add them to a playlist. For instance, below are screenshots of a YouTube playlist with four YouTube videos (left), and a YouTube Music playlist with four songs (right). I could add more songs or videos to the playlists and remove and reorder items as well.


pl 0
PLAY
SHUFFLE
4 videos · Sort
What is SMTP? and how it works? the Simple...
H-EDUCATE
What is SMTP (Simple Mail Transfer Protocol)?
Eye on Tech
E-Mail Protocols (SMTP, POP and IMAP)
Rick Graziani
SMTP | SMTP Protocol | Simple Mail Transfer...
Chirag Bhalodia
pl 0
Private · 4 songs
SHUFFLE
PLAY
Cello Suite No. 1 in G Major, BWV 1007: I. Prélude
Pablo Casals · 2:28
Cello Suite No. 4 in E-Flat Major, BWV 1010: I. Prélude
Pablo Casals · 4:15
Cello Suite No. 2 in D Minor, BWV 1008: I. Prélude
Pablo Casals · 3:43
J.S. Bach: Suite For Cello Solo No.1 in G Major, BWV 1007 - 1. Prélude
János Starker · 2:31
Add a song
Suggestions

43. The YouTube application also permits users to transfer playback from the YouTube application to a device that supports "Casting." The image below shows the Cast icon in the YouTube application (left image). A user may select the Cast icon to view one or more devices that are available for casting (right image).


HITS 2020 - Hit List 2020 (Best New Hits Pla...
Switch to YouTube Music
ENJOY NOW
Topic, A7S - Breaking Me ft. A7S
SUBSCRIBE
Cast to
Chromecast Ultra
Hub Speaker

Sonos February 24, 2022 Supplemental Infringement Contentions, Ex. A at 24.

44. Sonos has also accused functionality in YouTube's "MDx," or "Multi-Device Experience" protocol. MDx is a protocol that governs sending and managing playback between a YouTube application and a playback device. The YouTube application and playback device are communicatively coupled to one another through their mutual connection to an MDx Server (which is also sometimes called a YouTube "Lounge Server"). Put another way, the YouTube

[2] In doing so, I do not agree that these products operate in materially the same way for purposes of infringement. For instance, the YouTube Kids and YouTube TV application do not allow users to create playlist or queues. I have not attempted to identify all material differences between these applications in this declaration.

application sends a message to a playback device by sending a message to the MDx server, which then transmits the message to the playback device. Similarly, a playback device sends a message to a YouTube application by sending a message to the MDx server which then forwards that message to the playback device.

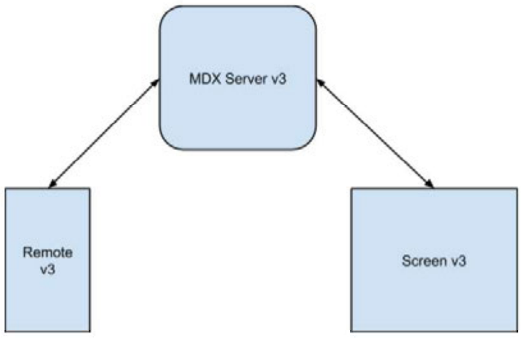
45. I provide additional details regarding the MDx protocol below.

(a) Google's MDx Protocol

46. As mentioned above, the protocol used by the YouTube System to control and manage playback between a YouTube Remote and YouTube Screen is called MDx.

47. I have reviewed the declaration of Ramona Bobohalma ("Bobohalma Decl."). I understand work on MDx Version 1 began no later than 2010, in connection with the prior art YT Remote system that was released in that same year. Bobohalma Decl., ¶3. The YT Remote and Version 1 of the MDx is prior art that I discuss later in this Declaration. I understand that by September 2011 Google had moved on to development of Version 2 of the MDx protocol. Bobohalma Decl., ¶3. I further understand that by 2014, Google had refined the protocol and created Version 3 of the MDx protocol. *Id.*

48. The accused YouTube applications use Version 3 of MDx. The MDx protocol provides that a YouTube application and a playback device (*e.g.*, a TV) are communicatively coupled through an MDx Server (also called a "Lounge Server"), as shown on the right. This general architecture of the MDx protocol has remained the same since Version 1. Ex. 2 at GOOG-SONOSWDTX-00041745.

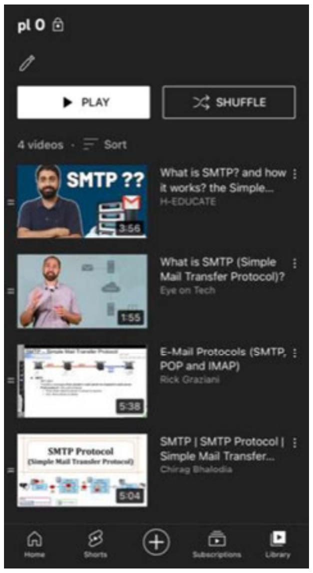


49. Whereas early versions of the MDx protocol—such as that used in the YouTube Remote prior art discussed in Section VII—maintained a playback queue on the Screen, in Version 3 Google changed MDx to eliminate the playback queue on the Screen in favor of maintaining it on the MDx server: "the queue is now maintained on the MDx server and not the TV." See, e.g., Ex. 2 [MDx Communication Protocol v3 Differences] at GOOG-SONOSWDTX-00041748; Ex.

3 [MDx Overview 20Q3] at GOOG-SONOSWDTX-00039988 ("MDx is the first server-backed Cloud queue at YT"). Google's documents explain that when a mobile device "Casts" (i.e., sends) playback to a playback device, the queue is stored in a "remote queue"—not a local queue. Ex. 4 [The Queue] at GOOG-SONOSWDTX-00039798 ("[w]hen Casting, the queue is persisted as a server-side 'remote queue'"), GOOG-SONOSWDTX-00039799 ("Casting use case stores the queue in YouTube servers as a "Remote Queue" playlist.), GOOG-SONOSWDTX-00039800 ("MDx Session Server manages the 'Remote Queue' playlist," and users "make queue edit operations (add to queue, re-order, remove from queue, etc.) through the MDx Session Server.").

50. Google's documents refer to the queue that is stored on the MDx server as a "Cloud Queue," "Shared Queue," "and "Remote Queue." Accordingly, I too will use these terms.

(b) Exemplary Walkthrough Of A Playback Transfer

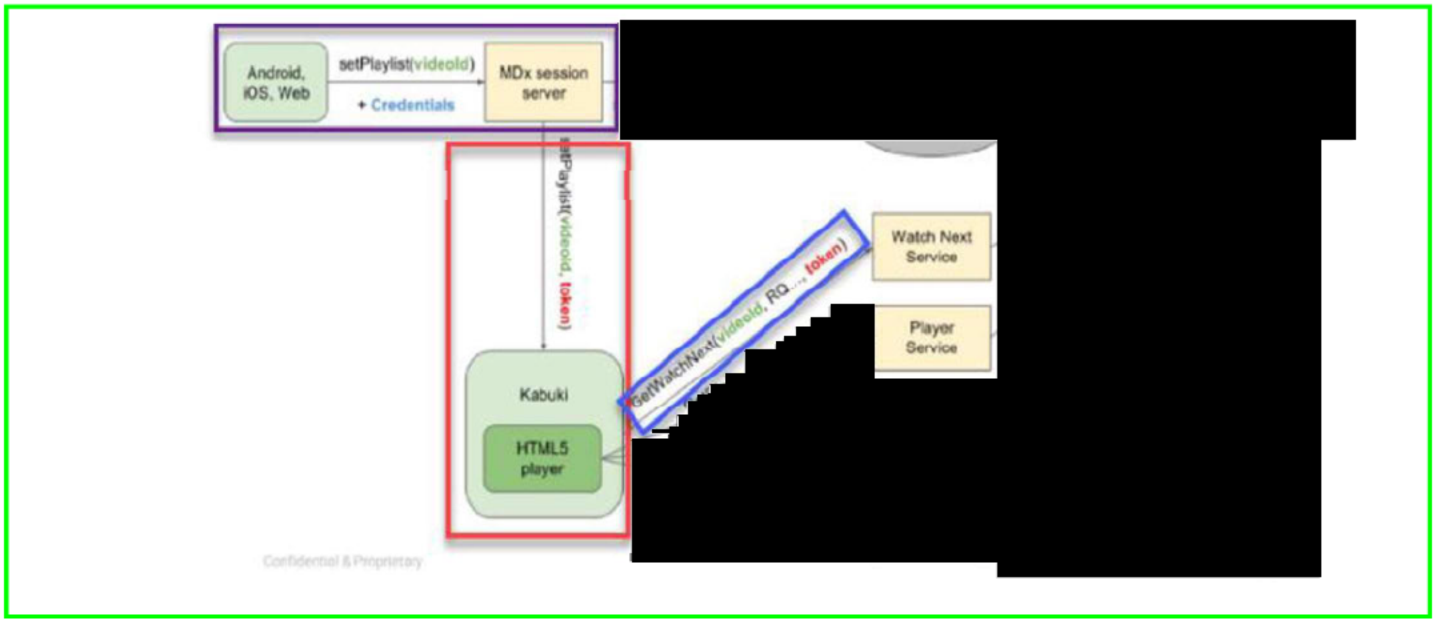

pl 0
PLAY
SHUFFLE
4 videos · Sort
SMTP ??
3:56
What is SMTP? and how it works? the Simple...
H-EDUCATE
1:55
What is SMTP (Simple Mail Transfer Protocol)?
Eye on Tech
5:38
E-Mail Protocols (SMTP, POP and IMAP)
Rick Graziani
SMTP Protocol
5:04
SMTP | SMTP Protocol | Simple Mail Transfer...
Chirag Bhalodia
Home
Shorts
Subscriptions
Library

51. I will now walk through the process by which a YouTube application transfers playback of a playlist of media to a playback device using the example playlist ("pl 0" on the right). One term I will use throughout this description is "videoId." A videoId in YouTube is simply a unique identifier for a video or song.

52. The Playlist is stored in the Shared Queue on the MDx server by sending an identifier for the playlist (referred to as a "listID," "playlistID," or "shared queue id") that causes the MDx server to store the playlist in a Shared Queue on the MDx server as an ordered list of videoIDs (along with Credential Transfer Tokens). Ex. 6 at GOOG-SONOSWDTX-00039491 (*see* "How a Playlist Gets Added to the Shared Queue."). I discuss the properties of the Shared Queue in greater detail later in this declaration (*see* ¶65).

53. I will now describe some of the steps that are taken when media is transferred and played back by the playback device. A user may begin playing the first media item in the playlist—"What is SMTP…"—on their device (represented by the "Android, iOS, Web" box above) and then send playback of the video to the playback device (represented by the "Kabuki" box). In this

example I will assume a single playback device was selected for transfer. I will also refer to the colored boxes I have added to the image below:



[redacted]

Ex. 6 at GOOG-SONOSWDTX-00039491.

54. As shown in the purple box in the image above, transferring playback causes the YouTube application to send a "setPlaylist" message to the MDx server which contains, among other things, the videoID of the video (the video "What is SMTP") and a playlistID for the playlist (playlist "pl 0"). In particular, when a user sends playback of a media item to a playback device,

[redacted]

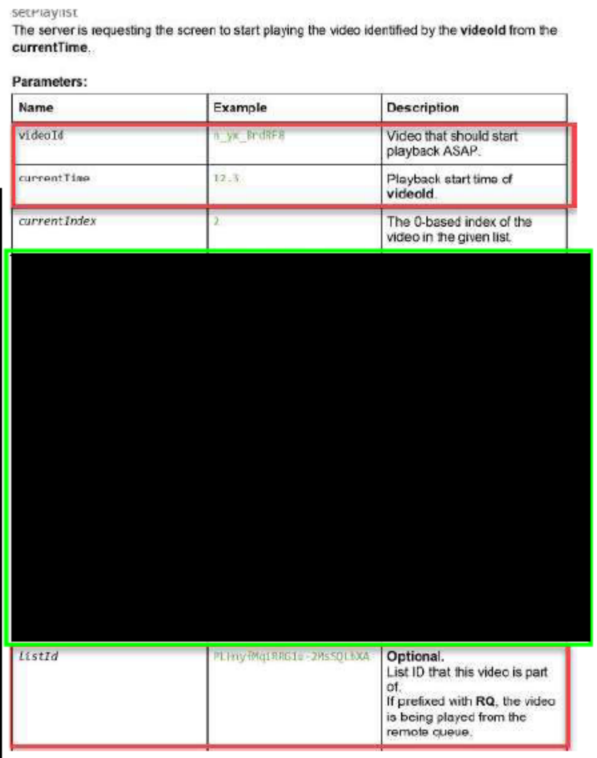
55. As shown in the red box in the image above, after receiving the setPlaylist message from the YouTube application and storing the playlist in the MDx server, the MDx server sends a setPlaylist message to the playback device. The setPlaylist message sent from the MDx server to the playback device includes the videoId of the media item that should start playback—in our

[3]In 2020-09-22-youtube_android_15.38.35/java/com/google/android/libraries/youtube/mdx/remote/internal/

example the video "[redacted]"—[redacted]

[redacted] the playlist "pl 0"[redacted]

[redacted]



setPlaylist

The server is requesting the screen to start playing the video identified by the **videoId** from the **currentTime**.

Parameters:

Name	Example	Description
videoId	n_yx_BrdRF8	Video that should start playback ASAP.
currentTime	12.3	Playback start time of **videoId**.
currentIndex	2	The 0-based index of the video in the given list.
[redacted]	[redacted]	[redacted]
listId	PLhny7Mq1RRG1e-2MsSQLbXA	**Optional.** List ID that this video is part of. If prefixed with **RQ**, the video is being played from the remote queue.

[redacted] Ex. 5 [MDx Communication Protocol v3] at -00037251-252

57. Continuing to refer to the red box in the image of the accused system above (¶53),

[redacted]

[4] In 2021-02-01_YTServerMDx09292020/google3/java/com/google/youtube/lounge/browserchannel/

[5] In 2021-02-01_YTReceivers09292020/google3/video/youtube/src/web/javascript/library/mdx/screen_ts

[6] In 2021-02-01_YTReceivers09292020/google3/video/youtube/tv/bedrock/ts/mdx/services

[redacted]

[redacted] (remote.ts, line 123). [redacted] Indeed, the variables are used in remote.ts to perform, by way of example, various book-keeping functions and to report information back to the lounge server (*e.g.*, to see whether likes are available for the video (line 2568), and to notify other system components what video is playing (line 851)).

58. Turning to the blue box in the image above (¶53), to play the next video, the playback device sends a WatchNext request to YouTube's InnerTube service. [redacted]

59. After sending the WatchNext request, the playback device receives a WatchNext response that contains a variety of information. For example, Sonos has pointed to

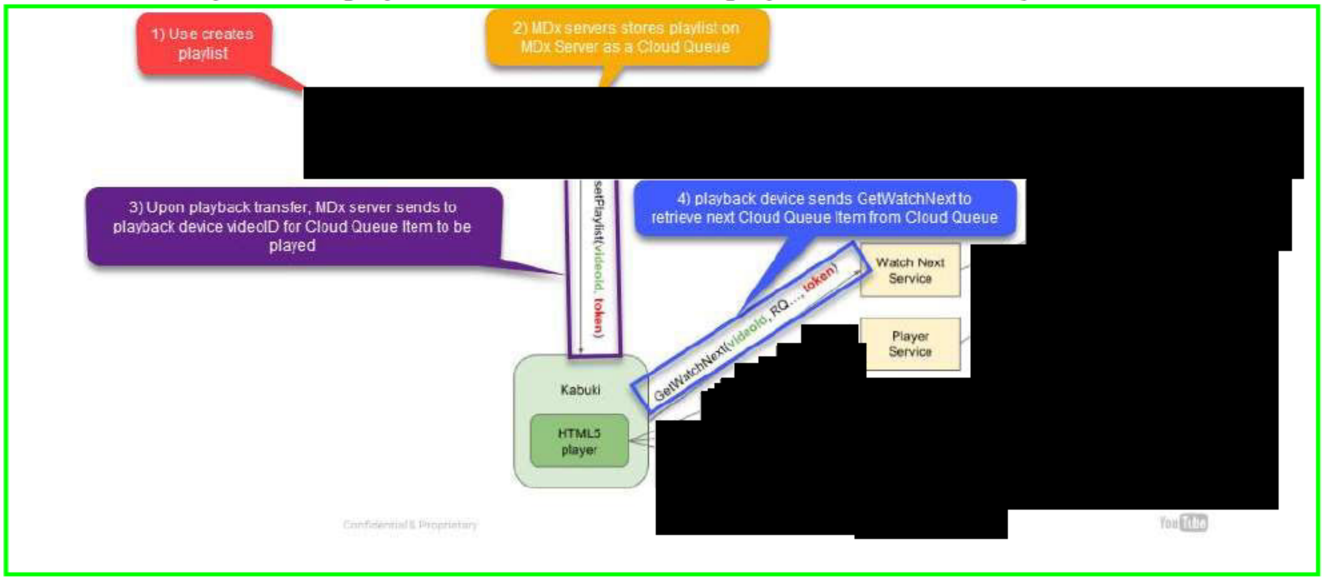
[7] In 2021-02-02_YTServerInnerTubeWatchNext09292020/google3/video/youtube/api/innertube/proto/

[8] In 2021-02-02_YTServerInnerTubeWatchNext09292020/google3/video/youtube/src/python/servers/innertube/watch_next/

[9] In 2021-02-02_YTServerInnerTubeWatchNext09292020/google3/video/youtube/src/python/servers/innertube/watch_next/

watch_next_response.ts[10] in its infringement contentions. The file that Sonos points to references information that may be included in a WatchNextResponse. For example, it references "WatchNextAlertsSupportedRenderers" (line 20) that includes information about the current video that is playing, UI information for alerts that are overlaid on the screen (alert_renderer.proto[11]), "contextual menus to show over a video" that "can contain actions such as liking or dis-linking a video" (player_overlay_renderer.proto[12]), "text describing the number of subscribers for the channel," "text describing the number of videos on the channel," "badges associated with the channel" (video_owner_renderer.proto[13]), etc. Within this large volume of information in the WatchNextResponse is also the videoId for the next media item in the Cloud Queue that the YouTube Screen should play. Each time a playback device moves to the next media item in the Cloud Queue, it makes a WatchNext request.

60. Below, is an annotated graphic that provides a high-level summary of the steps that I discussed by which a playback device receives and plays back the Cloud Queue:


1) Use creates playlist
2) MDx servers stores playlist on MDx Server as a Cloud Queue
3) Upon playback transfer, MDx server sends to playback device videoID for Cloud Queue Item to be played
setPlaylist(videoId, token)
4) playback device sends GetWatchNext to retrieve next Cloud Queue Item from Cloud Queue
GetWatchNext(videoId, RQ..., token)
Watch Next Service
Player Service
Kabuki
HTML5 player
Confidential & Proprietary
YouTube

[redacted]

Ex. 6 at GOOG-SONOSWDTX-00039491.

[10] In /2021-02-01_YTReceivers09292020/google3/video/youtube/tv/bedrock/ts/innertube/models

[11] In 2021-02-02_YTServerInnerTubeWatchNext09292020/google3/video/youtube/api/innertube/proto/renderers/

[12] In 2021-02-02_YTServerInnerTubeWatchNext09292020/google3/video/youtube/api/innertube/proto/renderers/

[13] In 2021-02-02_YTServerInnerTubeWatchNext09292020/google3/video/youtube/api/innertube/proto/renderers/sections

2. The Accused YouTube Applications Do Not Satisfy At Least The "Multimedia Content," "Local Playback Queue" And "Resource Locator" Limitations

61. Claim 13 of the '615 patent recite a step of transferring playback from a control device to a playback device causes, among other things, "causing one or more first cloud server to ***add multimedia content to a local playback queue on the particular playback device***," (the "multimedia content" and "local playback queue" limitations), and further recites that "adding the multimedia content to the local playback queue comprises… ***adding, to the local playback queue, one or more resource locators corresponding to respective locations of the multimedia content at one or more second cloud servers***" (the "resource locator" limitation). '615 patent, Claim 13.

62. In my opinion, the accused YouTube application do not infringe for at least three, independent reasons. <u>First</u>, YouTube does not use a "local playback queue on the particular playback device"—it uses a playback queue that is stored remotely in the cloud (a "Cloud Queue"). <u>Second</u>, YouTube does not "add multimedia content" to the non-existent "local playback queue." <u>Third</u>, YouTube does not add "one or more resource locators corresponding to respective locations of the multimedia content at one or more second cloud servers" to the non-existent local playback queue. I discuss each of these missing limitations below.

(a) YouTube Does Not Infringe Because It Uses A Cloud Queue, Not A "Local Playback Queue"

63. The "local playback queue" limitation requires that the step of transferring playback from the alleged "control device" to the alleged "playback device" causes one or more first cloud server to "add multimedia content to ***a local playback queue*** ***<u>on the particular playback device</u>*.**" Sonos contends that a "smartphone, tablet, or computer device" that runs the accused YouTube applications is the claimed "control device," and that "Cast-enabled media player," such as a Google speaker or TV is a "playback device." *See* Sonos '615 Claim Chart at p. 1-3. In other words, to satisfy this limitation the "playback queue" must be stored locally, on the Cast-enabled media player. A playback queue that is stored remote to the Cast-enabled media player, such as on the alleged control device or on a cloud server would not satisfy the claim.

64. It is my opinion that Google's accused YouTube system does not satisfy the "local

playback queue" limitation under Google's proposed construction of "playback queue" or any reasonable interpretation of the term. In the accused YouTube system, the playback queue is not stored on a playback device, such as the playback device. Rather, as already mentioned, it is stored in the Cloud Queue. *See, e.g.*, supra ¶49. Indeed, the protocol used by the accused YouTube system for controlling playback is Version 3 of the MDx protocol, which was changed from earlier version of MDx—such as Version 1 that is used in the prior art YT Remote discussed in Section VII—to eliminate the "local playback queue" in favor of a Cloud Queue.

65. The source code for the accused YouTube system also confirms that the accused YouTube system stores the queue remotely on the MDx server, not locally on the playback device. This Cloud Queue on the MDx server has the characteristics that a POSITA would attribute to a playback queue, including that it is an ordered list and can be managed by a user:

> **<u>Ordered List.</u>** The Cloud Queue in the YouTube system includes an ordered list of multimedia items. For example, I have reproduced lines 28-35 of ShareQueue.java below.
>
> [redacted]
>
> **<u>Ability To Manage Queue (Add, Remove, And Rearrange Items In The Queue).</u>** A POSITA would understand that a queue can be edited and managed, with items added, moved or removed from the queue. The Cloud Queue can also be edited and managed.
>
> [redacted]

66. I understand that Sonos has presented three different theories for how an alleged playback device stores a "local playback queue":

> Sonos contends that that (i) each of Google's data variables currentVideoIdDeprecated and currentWatchEndPoint.videoID amounts to the claimed "local playback queue" with the "videoID" of the current media item amounting to "an identifier of the multimedia content," (ii) alternatively, one or both of Google's data variables currentVideoIdDeprecated and currentWatchEndPoint.videoID in combination with Google's data variable upNextVideoID amounts to the claimed "local playback queue"

14 [redacted]

> with the "videoID" of the current media item and the "videoID" of the next media item each amounting to "an identifier of the multimedia content," and (iii) alternatively, Google's "WatchNextResponse" data structure by itself (or in combination with one or more of the aforementioned data variables) amounts to the claimed "local playback queue" with the "videoID" of one or more of the current, previous, and/or next media item amounting to "an identifier of the multimedia content."

Sonos's 2-24-2022 Supplemental Infringement Contention, Ex. A at 41-42. But these theories merely point to implementation details of YouTube's Cloud Queue—not a "local playback queue on the particular playback device." I discuss each of Sonos's theories below an demonstrate that they are not a "local playback queue" under Google's proposed construction, or any reasonable interpretation of "local playback queue."

(i) Sonos's First And Second Theory Fail To Identify A "Local Playback Queue On The Particular Playback Device"

67. Sonos's first theory is that each of the variables currentVideoIdDeprecated or currentWatchEndPoint.videoID amount to the claimed "local playback queue." Sonos's second theory points to these variables in conjunction with the variable upNextVideoID. In my opinion, neither of these theories show a "local playback queue," whether under Google's proposed construction or any reasonable interpretation of the term "local playback queue."

68. As reflected in Google's construction, a "playback queue" is an "ordered list of multimedia items selected by the user for playback." Thus, under Google's proposed construction a "local playback queue on a particular playback device" is an "ordered list of multimedia items selected by the user for playback" that is stored within the particular playback device. In YouTube, the ordered list of multimedia items selected by the user for playback (*e.g.*, the playlist "pl 0" show above) is stored in the Cloud Queue, as I previously explained. *See* Section VI.A.1(b).

69. Further, the accused variables cannot be a "local playback queue on the particular playback device" under any reasonable interpretation of that term.

70. A POSITA would understand that typical implementation of a playback queue links together different multimedia items in a particular order using linked lists, arrays, vectors, or other well-known data structures. Any implementation of a queue would allow for the storage of multiple items. A POSITA would not consider a fixed set of named variables to be a playback

"queue." For instance, Thomas H. Cormen's book Introduction to Algorithms (available here) includes a chapter describing "Elementary Data Structures," which describes "Queues." Chapter 10. The books disclose exemplary implementations of a Queue (*e.g.,* an array or a linked list), each of which is a structure that can store items in a particular order. *Id.* Tellingly, there is no example using a fixed set of named variables stored in memory.

71. The accused variables here can store only a single item at a given time. In particular, currentVideoIdDeprecated and currentWatchEndPoint.videoID are the same thing: a variable that stores the videoID for the media item that is currently being played on the playback device ("the current videoID").[15] Similarly, after the playlist in the Cloud Queue has been exhausted, the YouTube servers may send the playback device a videoID for a recommended "autoplay" video that is stored in the upNextVideoID variable. In other words, Sonos has at most pointed to variables that stores a single item at a given time—not a queue.

72. Further, as Sonos's expert, Dr. Schmidt, explained in his declaration "a 'playback queue' (or 'play queue') can have 'zero, one, or multiple media items at a given time.'" Schmidt Decl., ¶¶52, 84 (citing US Patent Application Publication 2012/0089910). In contrast, as just mentioned, the variables that Sonos accuses can each hold only one media item (and collectively, two media items). This is inconsistent with not only the plain meaning of "local playback queue," but also YouTube, which allow users to create playlists with many more items. For instance, the playlist "pl 0" I discussed above includes four media items, and could include many more. *See supra*, ¶51. Tellingly, the Cloud Queue is able to store all the media items in the playlist.

73. Users may edit or manage a queue, including by reordering the queue, adding to the queue, removing from the queue, shuffling, and so on. Indeed, the '615 patent discloses that a queue is something the user maybe "editing/managing." '615 patent at 16:25-31 ("Two-way

[15] I understand that the currentWatchEndPoint.videoID is a replacement for currentVideoIdDeprecated. [redacted]

communication helps enable features such as keeping a local playback queue synchronized with a queue that the user is editing/managing in a third-party application.”). Similarly, the Cloud Queue stored on the remote MDx server and the “music queue” in Sonos’s own devices at the time of the alleged invention also supported the ability to edit and manage the playback queue, as I have shown in the table below. The variables Sonos accuses are not capable of this type of queue management.

<u>Sonos User Manual</u> (Dkt. No. 200-12)	<u>Cloud Queue On MDx Server</u>
Queue is an ordered list of tracks. *See* Page 4-2 (“What is a music queue? When you make music selections, they are added to a list of tracks called a music queue.”).	Queue is an ordered list of media items. *See* SharedQueue.java, line 35.
Tracks can be removed from the queue. *See* Page 4-10 (“Managing the Music Queue… Removing a track from the queue.”)	Media items can be removed from the queue. *See* SharedQueue.java, line 56.
Tracks can be moved within the queue. *See* Page 4-11 (“Managing the Music Queue… Moving a track within the queue.”)	Media items can be moved within the queue. *See* SharedQueue.java, line 51.
Music queue can be cleared. *See* Page 4-11 (“Managing the Music Queue… Clearing the music queue.”)	Queue can be cleared. *See* SharedQueue.java, line 61.

74. That the accused YouTube system does not use a “local playback queue” is further evidenced by the fact that in the accused system if the MDx servers were to go offline, playback of the playlist would not be possible. In contrast, in a system that stored the “playback queue” locally on the device, the playback device could continue to play the queue—whether a remote copy was available or not. As an example, consider the prior art YouTube Remote system that I discuss in this Declaration (Section VII): the playback device in the YouTube Remote prior art stores a list of videoIDs for the playlist and is thus capable of playing back the playlist even if the MDx server were not available.

75. Sonos’s first and second theories fail for additional reasons as well.

76. For instance, Sonos’s first theory fails because Sonos has not pointed to any source code showing that the currentVideoIdDeprecated and currentWatchEndPoint.videoID variables are actually used in the playback process. Indeed, in remote.ts (the file Sonos points to in its

infringement contentions) these variables are used to perform various book-keeping functions and to report information back to the lounge server (*supra*, ¶57). This further confirms that these variables cannot form a "*playback* queue."

77. Sonos's secondary theory fails for multiple additional reasons. First, the upNextVideoID variable that Sonos has accused is a local variable in the handleWatchNextLoaded() function (remote.ts, line 1721-1754). Because it is a local variable within a function, it is only accessible while that function is executing. [redacted] In my opinion, a variable that exists for a few microseconds and is not used to commence playback cannot form part of a playback queue.

78. Further, the videoID for the recommended video is provided to the playback device only *after the playlist is exhausted*. Thus, the videoID that is stored in upNextVideoID for a few microseconds is not part of the playlist and is not "selected by the user for playback," as required by Google's proposed construction of "playback queue."

79. Additionally, Claim 13 recites that the step of transferring playback must cause items to be added to the "local playback queue," *see* supra, ¶13, such as "resource locators" and "multimedia content." This means that the local playback queue must exist as part of the transferring playback step. The videoID stored in upNextVideoID, however, is only used after the playback device has exhausted playback of the playlist, which would only occur after the transferring playback step is complete.

(ii) Sonos's Third Theory Does Not Identify A "Local Playback Queue On The Particular Playback Device"

80. Sonos's third theory contends that "Google's "WatchNextResponse" data structure" amounts to a "local playback queue." A POSITA would not consider a WatchNextResponse to be a "playback queue" under Google's proposed construction or any reasonable interpretation of the term for many reasons.

81. Each time a playback device moves to the next media item in the Cloud Queue, it makes a WatchNext request and receives a WatchNext response. (*see* supra, ¶59). The WatchNextResponse file that Sonos points to references information that may be included in a WatchNextResponse. For example, it references "WatchNextAlertsSupportedRenderers" (line 20) that includes information about the current video that is playing, UI information for alerts that are overlaid on the screen (alert_renderer.proto), "contextual menus to show over a video" that "can contain actions such as liking or dis-linking a video" (player _overlay_renderer.proto), "text describing the number of subscribers for the channel," "text describing the number of videos on the channel," "badges associated with the channel" (video_owner_renderer.proto), etc.

82. Sonos has not explained how this varied set of information is a "playback queue." Indeed, as I explained previously, a POSITA would understand that a playback queue is stored in a data structure, and that a typical implementation of a queue links together different multimedia items in a particular order using linked lists, arrays, vectors, or other -known data structures.

83. At most, Sonos has shown that this large volume of information contains within it a videoID corresponding to the next item in the Cloud Queue that should be played. But the fact that the playback device must send and receive a WatchNext message each time it moves to the next media item in the Cloud Queue reflects the use of a Cloud Queue—not a "local playback queue." If the playback queue were stored locally on the playback device, it would be unnecessary to retrieve items from the Cloud Queue one-by-one.

84. Further, as I explained earlier, a "playback queue" can be managed and edited, for example by adding items to the queue, removing items from the queue, shuffling the playback order and so on. *See, e.g.*, ¶¶21, 73. The WatchNextResponse does not support any of these types of queue management functions. In fact, Sonos's purported "playback queue" only persists for the length it takes to playback a single media item because a new WatchNextResponse is retrieved each time the next item in the Cloud Queue is played. A POSITA would understand that a playback queue persists beyond the length of a single playback.

(iii) Sonos Has Not Shown The Local Playback Queue Limitation Is Met Under The Doctrine Of Equivalents

85. I understand that Sonos contends that "even under Google's theory that a combination of multiple, individual data variables does not literally amount to a 'local playback queue on the particular playback device' the combination of the currentVideoIdDeprecated, currentWatchEndPoint.videoID, and upNextVideoID would infringe "under DoE." I disagree.

86. The combination of variables that Sonos points to are substantially different than a "local playback queue." These variables store a single Cloud Queue item from a Cloud Queue. They do not store a "local playback queue." A cloud queue provides centralized storage of the playback queue. As a result, it can support playback and interactions from a large number of users and can be shared with many devices. A cloud queue is not restrained by the capabilities of the playback device's hardware (*e.g.*, memory), and the queue is not lost when a playback device fails.

87. I understand that Sonos has alleged that the accused variables and the claimed "local playback queue" perform substantially the same function by "maintaining two or more multimedia items for playback." I disagree that this broad characterization shows the "local playback queue" is substantially the same as the variables Sonos is accusing. The "local playback queue" maintains the "playback queue" (*i.e.*, the ordered list of multimedia items selected by the user for playback) locally on the playback device. The individual variables Sonos accuses are unable to maintain a playback queue locally. They can only maintain one Cloud Queue item at a given time (and collectively two cloud queue items).

88. In addition, the accused variables and the claimed "local playback queue" do not perform the function in substantially the same way. The "local playback queue" stores a playback queue locally on the device. In contrast, the data variables Sonos point to store individual Cloud Queue items, while the actual playback queue is stored remotely in the Cloud Queue.

89. Nor is it the case that the accused variables and the claimed "local playback queue" achieve substantially the same result. The local playback queue allows the playback device to playback the playback queue without the need for a Cloud Queue. Because the variables Sonos accuses can store only a single item from the Cloud Queue (and collectively two items), they

cannot playback the playback queue without a Cloud Queue. If the Cloud Queue were to become unavailable, the local playback queue would continue to playback the multimedia content, whereas the individual variables Sonos points to would not.

(b) YouTube Does Not Infringe Because It Does Not "Add Multimedia Content" To Any Alleged Queue

90. The "multimedia limitation" requires that the step of transferring playback "add multimedia content to a local playback queue." As I discussed above, Sonos identifies the local playback queue as the variables currentVideoIdDeprecated, currentWatchEndPoint.videoID, upNextVideoID, or the WatchNext Response. *Supra*, ¶66. Thus, Sonos must identify "multimedia content" that is added to these elements. Sonos has not done so.

91. None of these structures store "multimedia content." At most, Sonos has shown that they may store a videoId. But a videoId is merely a unique identifier for the content and is typically only a few characters long, for example the videoID may be "n_yx_BrdRF8." An identifier of multimedia is not the "multimedia ***content***." Rather, a POSITA would understand that "content" is the actual multimedia file or information that is played back.

92. The claims confirm that an identifier for multimedia (*e.g.*, a videoID) is not the "multimedia content." For instance, Claim 20 recites "causing an identifier of the multimedia content to be added to the playback queue," thereby distinguishing an identifier of the multimedia content from the multimedia content itself. Similarly, Claim 13 recites "playing back the retrieved multimedia content." A POSITA would understand that a videoId cannot be played back and thus cannot be the claimed multimedia content.

93. The specification of the '615 patent also refers to the "multimedia content" as the actual music or video, and distinguishes it from an identifier for the content which may also be stored in the queue. '615 patent at 12:58-67 ("Songs and/or other multimedia can be retrieved from the Internet"), 13:54-57 ("play music, audio, video and/or other multimedia content.").

94. Thus, in my opinion the accused YouTube System does not infringe because Sonos has failed to show that "multimedia content" is added to the variables and WatchNextResponse it

has identified as the alleged "local playback queue."

(c) YouTube Does Not Infringe Because "Resource Locators" Are Not Added To The Alleged Local Playback Queue

95. The "resource locator" limitation further requires that the cloud servers "add[] to the local playback queue, one or more resource locators corresponding to respective locations of the multimedia content at one or more second cloud servers." As I discussed above, Sonos identifies the local playback queue as the variables currentVideoIdDeprecated, currentWatchEndPoint.videoID, upNextVideoID, or the WatchNext Response. *Supra*, ¶66. Thus, Sonos must identify "one or more resource locators" that are added to these elements.

96. Sonos's first theory is that the "resource locator" limitation is satisfied because the data variables and WatchNext response above store "videoIDs," which Sonos contends is the claimed "resource locators." I disagree. Consistent with Google's construction, a POSITA would understand that "resource locator" should be construed as an "address of a resource on the Internet." Indeed, the phrase "resource locator" is used in the '615 patent only as part of the larger phrase "uniform resource locator" (URL). A URL is an address of a resource on the Internet. A videoId, in contrast, is merely a string of characters (*e.g.*, "n_yx_BrdRF8") that is a unique identifier for a song or video. The videoId is not an address and does not provide any information about the location where the video is stored. Thus, the accused videoIds do not satisfy the "resource locators" limitation if the Court adopts Google's construction.

97. Even if the Court does not adopt Google's proposed construction, a videoId still cannot satisfy the "resource locator" limitation because the claim language states that the resource locator must "correspond[] to respective locations of the multimedia content at one or more second cloud servers." Because it is merely an identifier, the videoId says nothing about where the media content is located, and the server that stores the content cannot be identified by the videoId. In fact, in the accused systems after a videoId is received it must subsequently be mapped to a server from which to request the content. The same videoId may be mapped to different servers depending on various conditions and circumstances. In other words, the location of the multimedia

contention from which the alleged playback device should retrieve the content is not even known at the time the videoId is received.

98. Sonos's second theory is that a URL retrieved by the using the videoId is a "resource locator." But the accused URLs cannot satisfy the "resource locator" limitation because it is not added to any of the items that Sonos identified as the alleged "local playback queue." Put another way, the accused variables and WatchNextResponse that Sonos has pointed to do ***not*** store any URLs. Thus, Sonos has failed to show that "resource locators" are added to the variables and WatchNextResponse it has identified as the "local playback queue."

B. The Accused Google Play Music Application Does Not Infringe Claim 13

1. Overview Of The Google Play Music Application

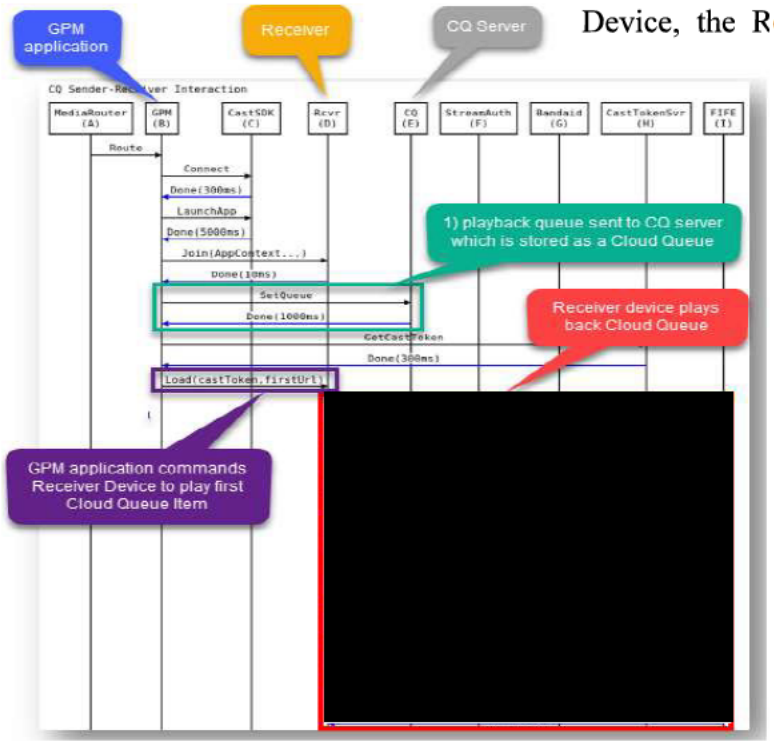
99. The Google Play Music ("GPM") application was a music application that provided music streaming services to users. I understand GPM has been discontinued.

100. By downloading and installing the GPM application on a mobile device that supports the application, users could view available music, add music to a playlist, and playback music on their mobile device. The GPM application also permitted users to send playback of a playlist from the mobile device to a Receiver Device, such as a speaker or TV through Google's "casting" feature ("Receiver Device"). I refer to the GPM application, Receiver device, and the remainder of the system as a "GPM System."

101. The cast feature was discussed above with respect to the accused YouTube system. The differences between casting in YouTube and GPM are not material to my opinions. Thus, I incorporate my discussion of casting in the YouTube applications section above.

102. Playback of music on a playback device was managed by Google's Cloud Queue API. In the GPM system, the queue was stored on a Cloud Queue Server, and a "Receiver Device" device played back items from the Cloud Queue.

103. In general, when a user sends playback from the mobile device to a Receiver Device, the Receiver Device plays items in the Cloud Queue by sending an "ItemWindow" request to retrieve a "window" that contains information about the previous, current and next Cloud Queue items. A general overview of this process is shown here with annotations added. Ex. 8 [Cloud Queue Casting Operations] at GOOG-SONOSWDTX-00043469.



[redacted]

104. As can be seen in the green box, [redacted] The Cloud Queue stored in the CQ servers contains a list of itemIds corresponding to the items in the Cloud Queue and further metadata about the queue. These include, for instance, the normal playback order of the items, whether shuffle mode is enabled, the shuffled playback order of the items, and playback modes (e.g., repeat track). *See* cloud_queue.proto[17], lines 303-327, 232-257.

105. Turning to the purple box, [redacted]

[16] In 2020-09-01-google_play_music_android_8.27.8839-3.U/java/com/google/android/libraries/play/music/playback2/players/

[17] In 2021-01-27_GPMServer09292020\google3\wireless\android\skyjam\cloudqueue\proto\

[18] In /2020-09-01-google_play_music_android_8.27.8839-3.U//java/com/google/android/libraries/play/music/playback2/remote/cast/

106. I note that the red annotation was initially provided by the plaintiff in its infringement contentions. I discuss that processing next.

107. In particular, [redacted] [redacted] [redacted] [redacted]). The ItemID is an identifier for the media item within the Cloud Queue that is to be played (the "CurrentCloudQueueItem").

108. The Receiver Device will begin playing the CurrentCloudQueueItem on the Receiver Device. The [redacted] The playCurrentAndFetchWindow function is shown, in part, below:

[redacted]

playermanager.js, line 2757-2770. As the comment in the source code explains (lines 2758-2759),

[redacted]

109. The playCurrent function (line 2737) is shown below and [redacted] playermanager.js, line 2730-2755.

19 In 2021-01-27_GPMReceiver09292020/google3/java/com/wireless/android/skyjam/frontend/javascript/cast/

110. [redacted] (number of upcoming tracks to fetch) to fetch items around the current item from the Cloud Queue (playermanager.js, line 1762-1814). [redacted]

111. [redacted]

112. Notably, I understand that beginning in 2013 the parties collaborated on integrating Sonos's speaker with GPM, including work on a Cloud Queue API for GPM. Google's Second Amended Complaint (Dkt. No. 125), ¶18. I have reviewed the Music Cloud Queue Rest API (GOOG-SONOSWDTX-00036998) and Cloud Queue Sequence Diagrams (GOOG-SONOSWDTX-00051100) both of which are labeled "Google/Sonos Confidential." The Cloud Queue functionality described in these documents is materially the same as that which is accused in this case. For example, the Music Cloud Queue Rest API discloses that the queue is stored in the Cloud Queue and that the playback device can retrieve a window of tracks from the Cloud Queue using an ItemWindow Request, including information for the previous ("previousWindowSize"), current ("ItemID"), and next ("upcomingWindowSize") media items.

[20] In 2021-01-27_GPMReceiver09292020\google3\java\com\google\wireless\android\skyjam\frontend\javascript\cast\cloudqueue\

GOOG-SONOSWDTX-00037000 [Rest API]; *Compare supra*, ¶¶108-111.

2. The Accused Google Play Music Application Does Not Satisfy At Least The "Local Playback Queue" Limitation

113. The GPM system does not infringe because it uses a Cloud Queue, not a "local playback queue on the particular playback device."

114. I understand that Sonos has presented the following theory for how the accused Google Play Music application stores a "local playback queue" on the YouTube Screen:

> In this regard, Sonos contends that Google's data structure "itemWindowResponse.items" amounts to the claimed "local playback queue" with each of the "itemID" and "trackUrl" of the "nextItem" amounting to the "one or more resource locators corresponding to respective locations of the multimedia content."

2-24 Supplemental Infringement Contention, Ex. A at 71. I disagree that itemWindowResponse is a "playback queue."

115. As proposed by Google, a "playback queue" is "an ordered list of multimedia items that is selected by the user for playback." In devices running the accused GPM application the ordered list of multimedia items that is selected by the user for playback is stored in a Cloud Queue. *See, e.g.,* GOOG-SONOSWDTX-00041650 at 55. It is not stored locally on the client device running the Google application. Sonos's "local playback queue" theory merely points to data or entries in the Cloud Queue. While a client device may choose to retrieve and buffer (or cache) data related to the queue that is stored in the cloud to optimize playback (*e.g.,* the itemID and track URLs), that data is not the ordered list of multimedia items selected by the user for playback. Thus, Sonos's theory fail for at least the reason that Google maintains a cloud queue.

116. Sonos has also failed to show a "local playback queue" under any reasonable construction of the term "playback queue" for the same reason I discussed in connection with the accused YouTube application.

117. For instance, a POSITA would understand that queues are stored in data structures, for example by linking together different multimedia items in a particular order using pointers,

arrays, vectors, or other well-known data structures. Merely caching information for the previous, current and next song does not form a queue for the same reason I discussed in connection with the accused YouTube applications. (*see* supra ¶70)

118. Further, a playback queue can be populated with a variable number of items, including zero, one or many items (*see* supra ¶72), whereas the ItemWindowResponse that Sonos points to only stores the previous, current and next items. The GPM application allows user to create playback queues that far exceed three items, further evidencing that the ItemWindowResponse is not a local playback queue.

119. As yet another example, a playback queue can be managed and edited, with items added to the queue, removed from the queue, reordered, shuffled, and so on. *See* supra ¶73. While the Cloud Queue functionality in GPM permits this type of queue management functionality, the ItemWindowResponse does not.

120. The fact that the playback device sends and receives an ItemWindowResponse each time it moves to the next media item in the Cloud Queue (*see supra* ¶111) also reflects the use of a Cloud Queue—not a "local playback queue." If the playback queue were stored locally on the playback device, it would be unnecessary to retrieve items from the Cloud Queue one-by-one.

VII. INVALIDITY ANALYSIS

A. Claim 13 Is Invalid Based On The YouTube Remote Prior Art.

121. I understand that on November 9, 2010, Google released Version 1.0 of its YouTube Remote application ("YTR application" or "YT Remote application"). Bobohalma Decl., ¶3; *see also infra*, ¶125. I understand that Google continued to update and released additional versions of the YTR application until the application was discontinued. *Id.*

122. Google also filed U.S. Patent No. 9,490,998 ("the '998 patent") on March 7, 2011, which claims priority to an earlier provisional application filed on November 8, 2010. I understand the '998 patent is based on the YouTube Remote. Bobohalma Decl., ¶4.

123. In my opinion, Claim 13 is anticipated based on the YTR system. To the extent the YTR system is deemed not to anticipate Claim 13, then Claim 13 is obvious based on the YTR

system in view of (1) the general knowledge of a POSITA and/or (2) the '998 patent. Below, I analyze each reference and show that they render Claim 13 invalid.

1. The Prior Art YT Remote System

(a) YT Remote Was Publicly Available By November 9, 2010.

124. I understand that YT Remote is prior art because it was publicly available before Sonos's alleged invention date (July 15, 2011). I understand that work on YT Remote began by mid-2010 and that the YTR application was released on November 9, 2010. Bobohalma Decl., ¶3.

125. Below I have listed videos that were uploaded to YouTube days after Version 1.0 of the YTR application was released, and a video that was uploaded in February of 2011. I understand that the below videos accurately describe the operation of the YTR application that was released to the Android Market on November 9, 2010 (Bobohalma Decl., ¶5), and Levai Decl., ¶5) and their operation is also consistent with my review of the source code that I understand corresponds to this release (Levai Decl., ¶6)." For ease of reference and citation, I have numbered each video and will use that video numbers in my analysis below:

- **Video #1**, GOOG-SONOS-WDTX-INV-00015101): How to control Google TV or YouTube Leanback with YouTube Remote, available at https://www.youtube.com/watch?v=EGdsOslqG2s (uploaded 11/14/210)
- **Video #2**, GOOG-SONOSNDCA-00071320): Youtube (Lean back) Remote App for Android, available at https://www.youtube.com/watch?v=5VFIuR9pJdo (uploaded 11/11/2010).
- **Video #3**, GOOG-SONOSNDCA-00071319): Samsung Galaxy Tab as Youtube Remote, available at https://www.youtube.com/watch?v=NR9jFT5MP6A (uploaded 11/9/2010)
- **Video #4**, GOOG-SONOS-WDTX-INV-00015102): YouTube Remote, available at https://www.youtube.com/watch?v=txIPVu6yngQ (uploaded 11/9/2010)
- **Video #5**, GOOG-SONOSNDCA-00071317): Android App "Youtube Remote," available at https://www.youtube.com/watch?v=lQXxYNCi2d4 (uploaded Nov. 15, 2010).
- **Video #6**, GOOG-SONOSNDCA-00071318): Android Tutorials – 14 Control YouTube with YouTube Remote, available at https://www.youtube.com/watch?v=Mnf9hKOc4Ro (uploaded 2/14/2011)

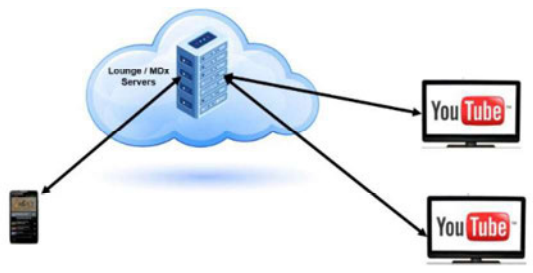
126. In order to understand the design and operation of the YTR system I have also

analyzed source code that I understand reflects the operation of the YTR system that was released on November 9, 2010. Levai Decl., ¶6. I have also analyzed the source code for the prior art YTR system as it existed prior to Sonos's alleged invention date of July 15, 2011. In particular, I analyzed the YTR system source code as it existed on July 12, 2011, which I understand is for subsequent releases of the YTR application. Levai Decl., ¶7. I have also analyzed the source code for the YTR system as it existed prior to the effective filing date of the '615 patent on December 30, 2011. In particular, I analyzed the YTR system source code as it existed on December 1, 2011. *Id.*

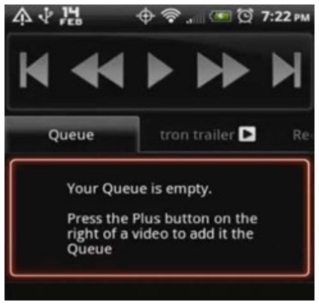
(b) Architecture And Operation Of The YT Remote System

127. The YTR application was designed for installation on a mobile device (*e.g.*, smartphones and tablets). It allowed users to view YouTube videos on their phone, create playlists of YouTube videos, and transfer playback of the playlist to one or more remote screens (*e.g.*, a TV or computer) running Google's "Leanback" application (I refer to a TV or computer running Leanback as a "Leanback Screen" or simply a "Screen"). When video playback was transferred from the YTR application on the mobile device to one or more Screens, the YTR application would serve as a remote control that could be used to control playback on the Screens.

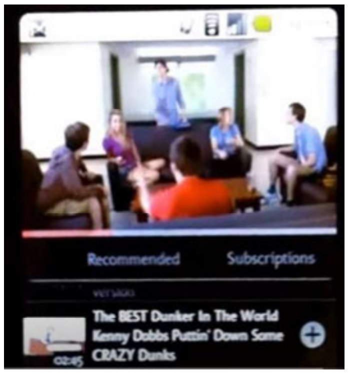
128. Like the accused YouTube applications, the YTR application and Screens were communicatively coupled to one another through their mutual connection to a YT Lounge Server. Put another way, the YTR application sends a message to a Screen by sending a message to the YT Lounge server (also called an MDx server), who then transmits the message to the Screen. Similarly, a Screen sends a message to a YT Remote application by sending a message to the YT Lounge Server who then forwards that message to the YT Remote application. On the right I have provided a simplified view of the basic architecture for the YTR system.



129. Using the YTR application, users could add videos to a queue. For example, the image on the right from **Video #6** shows a YTR application prompting a user to "press the plus button on the right of a video to add it to the queue." **Video #6** at 1:40-2:49 (showing user's queue on the YTR application); **Video #3** at 0:34-0:50 (showing user adding items to queue). Users could also edit the queue by, for instance, removing items from the queue or rearranging their order. **Video #6** at 2:43-2:2:58 ("if I hit edit, I can remove them from my queue or I can rearrange the order" of the queue).


7:22 PM
Queue
tron trailer
Your Queue is empty.
Press the Plus button on the right of a video to add it the Queue

130. The image on the right from **Video #1** shows the user interface of the YTR application when a user is playing a video directly on the mobile device. The YTR application permitted users to search for videos, and also provided them with a listing of recommended videos. A user of the YTR application could also rotate the phone, in which case the YTR application would play the video in full screen mode. *See* **Video** #1 at 0:25-0:40. The user could also play, pause, fast forward or rewind the video using the graphical interface on the YTR application. *Id*. at 1:33-2:00.


Recommended
Subscriptions
The BEST Dunker In The World Kenny Dobbs Puttin' Down Some CRAZY Dunks
02:45

131. I will now discuss the operations of the YTR system that relate to pairing the YTR application with Screens, transferring playback, and controlling playback after transfer.

(i) Pairing A YTR Application With The YT Lounge Server

132. A YTR application and one or more Screens are paired together by logging into the same YouTube account which registers them with the Lounge Server and adds them to a "session." Items 1 through 3 describe the general process by which a YTR application and one or more Screen are paired. GOOG-SONOSWDTX-00041837 ("remote tells the server a user has turned on their remote by registering a session…. Leanback registers with server in a similar manner to the remote… server sends a "loungeScreenConnected message to remote").

133. I have also confirmed that this functionality existed in the 2010 source code for Version 1.0 of the YT Remote that I reviewed. Incoming messages to the YTRemote application

are processed by CloudServerMessageListener.java[21] (processMessage method, line 38). When a Leanback Screen registers with the Lounge Server, the Lounger Server sends the YT Remote a loungeScreenConnected (CloudServerMessageListener.java, line 39) message. A lounge session may have multiple devices associated with it, in particular a single lounge session may support multiple Leanback Screens (LoungeSession.java[22], lines 25, 57).

134. The functionality also exists in the [redacted] that immediately precedes Sonos's alleged invention date. Just as in the 2010 code, [redacted] Again, a lounge session may have multiple devices associated with it, in particular a single lounge session may support multiple Leanback Screens (LoungeSession.java[24], lines 46, 204). In other words, the functionality in the 2011 and 2010 code remained the same, although the variable name "loungeScreen Connected" was changed to the name "SCREEN_CONNECTED."

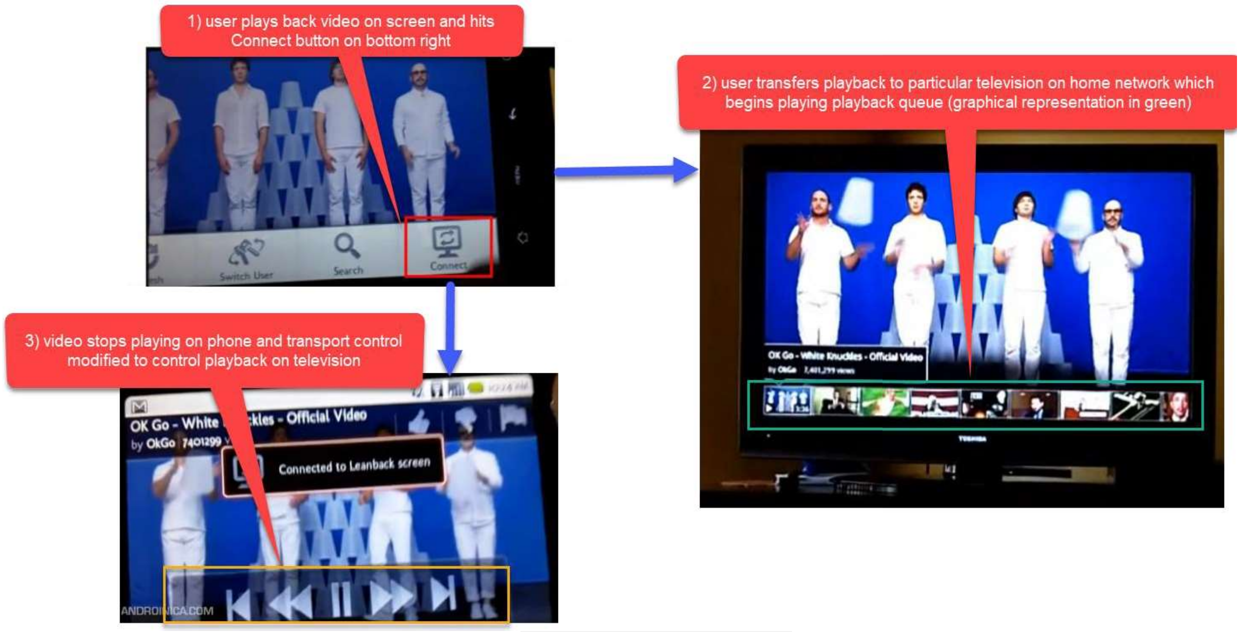
(ii) Transferring Playback Of Playlist From The YTR Application On The Mobile Device To The LeanBack Screen

135. The user may play YouTube videos on the YTR application from his or her queue, as I discussed above (*see* supra ¶¶129-130). The user may also transfer playback of the playlist from the mobile device to the Screen by tapping a "Connect" button on the YTR application.

136. The transfer playback functionality is shown in videos uploaded to YouTube. For instance, the images below are from a November 14, 2010 video. They show a user using the YTR application to play a YouTube video on their Android phone. The user thereafter taps the "menu"

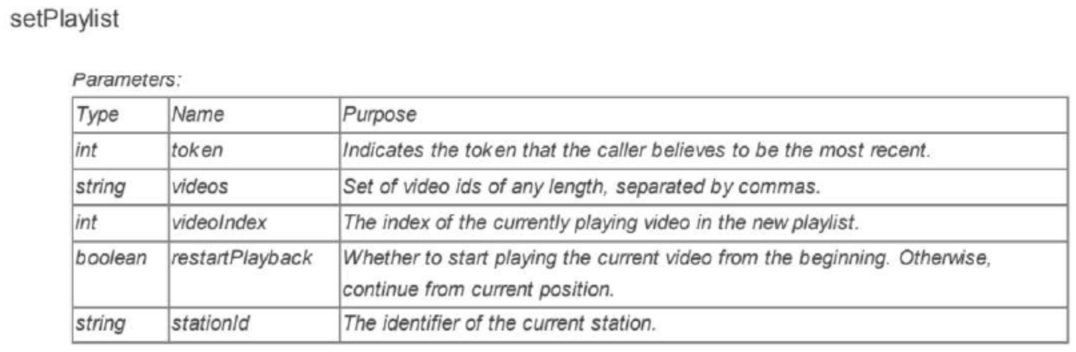
[21] In 2022-03-21_YTRemoteLeanbackAppsServer-11112010/google3/java/com/google/android/apps/ytlounge/src/com/google/android/ytremote/backend/
[22] In 2022-03-21_YTRemoteLeanbackAppsServer-11112010/google3/java/com/google/youtube/lounge/model/
[23] In 2022-03-22_YTRemoteLeanbackAppsServer07122011/google3/java/com/google/android/apps/ytlounge/src/com/google/android/ytremote/backend/
[24] In 2022-03-22_YTRemoteLeanbackAppsServer07122011/google3/java/com/google/youtube/lounge/model/

button on the Android phone to bring up a "Connect" button (shown on the top left image) and then by pushing that Connect button the playback of the video on the YT Remote application is stopped and resumes on the TV screen (shown on the right image). The YT Remote application is then used to control the Screen. I have annotated the images below to help explain the process:


1) user plays back video on screen and hits Connect button on bottom right
Switch User
Search
Connect
2) user transfers playback to particular television on home network which begins playing playback queue (graphical representation in green)
OK Go - White Knuckles - Official Video
3) video stops playing on phone and transport control modified to control playback on television
OK Go - White Knuckles - Official Video
by OkGo 7401299
Connected to Leanback screen
ANDROIDICA.COM

Video #1 at 1:05-2:00, (uploaded on 11/14/2010) at 1:00-1:30.

137. In particular, when the user pushes the "Connect" button to transfer playback, the YTR application sends a message to the Lounge Server with the current playlist. GOOG-SONOSWDTX-00041837 ("remote sends the current playlist to the screen."). More specifically, the YTR application sends a "setPlaylist" message to the Lounge Server, which the Lounge Server relays to the Screen. The contents of the setPlaylist message are described in the YT Remote API document excerpted below, and includes a string of identifiers for the videos in the playlist (called videoIDs) that are separated by commas (see row 3 of table below), as well as other information such as information about the currently playing video and the current position in the video:



setPlaylist

Parameters:

Type	*Name*	*Purpose*
int	*token*	*Indicates the token that the caller believes to be the most recent.*
string	*videos*	*Set of video ids of any length, separated by commas.*
int	*videoIndex*	*The index of the currently playing video in the new playlist.*
boolean	*restartPlayback*	*Whether to start playing the current video from the beginning. Otherwise, continue from current position.*
string	*stationId*	*The identifier of the current station.*

Ex. 10 (YT Remote API) at GOOG-SONOSNDCA-00056724); Levai Decl., ¶9.

138. I have also confirmed that this functionality existed in the 2010 source code for Version 1.0 of the YT Remote that I reviewed.

139. [redacted]

140. [redacted]

[25] in 2022-03-21_YTRemoteLeanbackAppsServer-11112010/google3/java/com/google/android/apps/ytlounge/src/com/google/android/ytremote

[redacted] In other words, the setPlaylist message is sent to all Screens that have been paired with the YTR application.

141. [redacted] This function creates a new local playlist (a "station") from the list of videoIds that are received (line 477). The playlist is stored locally as an array of variable size. (LeanbackModule.as, line 510-525, remotefeed.as, line 8, staticfeed.as, line 12, feed.as, line 22). [redacted] The subsequent processing uses a video player to retrieve the video. Different types of video players are supported by the devices on which Leanback operates. One example is the HTTPVideoPlayer (HTTPVideoPlayer.as[29]), which retrieves videos with URLs over the Internet. In particular, the videodata.videoURL variable stores the URL corresponding to the current media object to be played back. I understand that Screens used the playback URLs to retrieve videos from "Bandaid" servers on Google's Content Delivery Network. Bobohalma Decl., ¶3, Levai Decl., ¶3.

142. The functionality also exists in the July 12, 2011 source code that immediately precedes Sonos's alleged invention date.

143. The source code confirms that when a user presses the "Connect" button the video stops playing on the YTR Remote and a setPlaylist message is sent to the Lounge Server. In particular, when the user presses the connect button on the YTR application, [redacted]

[26] in 2022-03-21_YTRemoteLeanbackAppsServer-11112010/google3/java/com/google/youtube/lounge/browserchannel/

[27] in 2022-03-21_YTRemoteLeanbackAppsServer-11112010/google3/flash/as3/com/google/youtube/modules/leanback/

[28] in 2022-03-21_YTRemoteLeanbackAppsServer-11112010/google3/flash/as3/com/google/youtube/modules/leanback/

[29] in 2022-03-21_YTRemoteLeanbackAppsServer-11112010/google3/flash/as3/com/google/youtube/players/

[redacted] Similar to the documents, the source code for the YTR application [redacted]

144. [redacted] In other words, the setPlaylist message is sent to all Leanback Screens that have been paired with the YTR application.

145. [redacted] This function creates a new local playlist (a "station") from the list of videoIds that are received (LeanbackModule.as, line 670) which are stored as an array of variable size. [redacted] The subsequent processing uses a video player to retrieve the video. Many different types of video players are supported by the Screens on which Leanback operates. One example is the HTTPVideoPlayer (HTTPVideoPlayer.as[34]), which URLs to retrieve videos over the Internet. In particular, the

[30] in 2022-03-22_YTRemoteLeanbackAppsServer07122011/google3/java/com/google/android/apps/ytlounge/src/com/google/android/ytremote/

[31] in 2022-03-22_YTRemoteLeanbackAppsServer07122011/google3/java/com/google/android/apps/ytlounge/src/com/google/android/ytremote/

[32] in 2022-03-22_YTRemoteLeanbackAppsServer07122011/google3/java/com/google/youtube/lounge/browserchannel/

[33] in 2022-03-22_YTRemoteLeanbackAppsServer07122011/google3/flash/as3/com/google/youtube/modules/leanback/

[34] in 2022-03-22_YTRemoteLeanbackAppsServer07122011\google3\flash\as3\com\google\youtube\players\HTTPVideoPlayer.as"

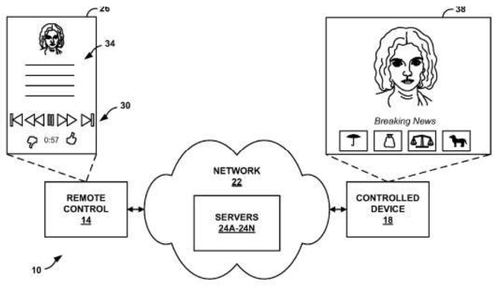
videodata.videoURL variable stores the URL corresponding to the current media object to be played back. I understand the videos retrieved by the playback URLs are stored on "Bandaid" servers in Google's Content Delivery Network. Bobohalma Decl., ¶3, Levai Decl., ¶3.

2. The Prior Art YT Remote Patent (The '998 patent).

146. U.S. Patent No. 9,490,998 Patent is titled "Network-based remote control." It was filed on March 7, 2011, and claims priority to November 8, 2010.

147. I understand that the '998 patent was based on the prior art YT Remote system. Bobohalma Decl., ¶4. The inventors of the '998 patent are Ramona Bobohalma, Daniel Danciu, Yaniv Bernstein, Oliver Heckmann, Jasmine Langridge, and Alin Sinpalean. At least some of these individuals (e.g., Ramona) are referred to as "authors" in the YT Remote source code.

148. The '998 patent discloses a web-enabled "remote control" (*e.g.*, a "cellular phone") that can "transmit control information via the network service (e.g., "servers 24") to" a "controlled device" (e.g., a "network enabled television") to control "playback of media content." '998 patent at 1:38-50, 3:14-55, 4:58-67, 5:1-12, 5:42-63, 5:64-6:13. Figure 1 (right) is an embodiment of the system.



149. The '998 patent discloses that the remote control may include a user interface, and that the user interface may display, for example, controls for playing back media (*e.g.,* "fast forward, reverse, skip ahead or back, play, stop, move to new content, etc."). '998 patent at 5:1-12. The controls and display information are shown as items 30 and 34 in Figure 1. "Remote controls and controlled devices may be paired using any one of several different techniques," for example by having the remote control and control devices login to the same "user account" and then notifying the server that they are "connected to the network" to initiate a "session." '998 patent at 4:21-32. Further, the '998 patent explains that in some embodiments the remote control application allows a user to select "one or more previously paired controlled device" to control: "user may also utilize the remote control application of remote control 75 to select one or more previously paired controlled devices." '998 patent at 10:67-11:6.

3. The YT Remote Anticipates Or Renders Obvious Claim 13 Alone, Or In View Of The YT Remote Patent ('998 Patent).

(i) Limitation 13 (preamble): "A tangible, non-transitory computer readable storage medium including instructions for execution by a processor, the instructions, when executed, cause a control device to implement a method comprising:"

150. To the extent the preamble is limiting, the prior art YT Remote system discloses the preamble in my opinion. For example, the YTR application runs on a mobile device, such as an Android smartphone or tablet, which each include a computer readable storage medium and a processor. The computer readable storage medium includes the software instructions that are executed by the processor to operate the YT Remote.

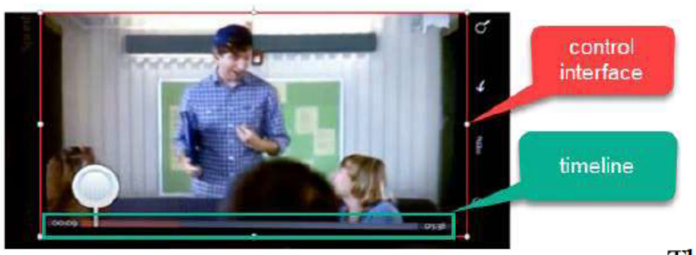
151. I understand that Sonos does not dispute that the YT Remote System discloses this limitation. Specifically, I understand that Google served an interrogatory requesting Sonos's contentions for why Claim 13 of the '615 patent is not invalid over the YT Remote system. I have reviewed Sonos's response ("Validity Contentions") as it relates to the YouTube Remote and Sonos does not dispute this limitation. *See* Validity Contentions at 184.

(ii) Limitation 13.1: "causing a graphical interface to display a control interface including one or more transport controls to control playback by the control device;"

152. In my opinion, the prior art YT Remote system discloses this limitation.

153. For example, the YT Remote System includes a "control device," for example a smartphone or tablet running the YTR application.

154. Further, the YTR application includes a graphical interface to display a control interface including one or more transport controls to control playback by the control device. The image on the left shows the control interface of the YTR application. The control interface includes transport controls (*e.g.*, forward, rewind, play, pause) to control playback by the control device. The user may fast forward or rewind the video by moving the circular white object across the timeline.



Video #1 at 0:33. A user could also play or pause a video by tapping on the screen. *Id.* at 1:03.

155. I understand that Sonos does not dispute that the YT Remote discloses this limitation. Specifically, I have reviewed Sonos's Validity Contentions as it relates to Claim 13 of the YouTube Remote and Sonos does not dispute this limitation. Validity Contentions at 184.

(iii) Limitation 13.2: "after connecting to a local area network via a network interface, identifying playback devices connected to the local area network;"

Limitation 13.3: "causing the graphical interface to display a selectable option for transferring playback from the control device;"

Limitation 13.4: "detecting a set of inputs to transfer playback from the control device to a particular playback device, wherein the set of inputs comprises: (i) a selection of the selectable option for transferring playback from the control device and (ii) a selection of the particular playback device from the identified playback devices connected to the local area network:"

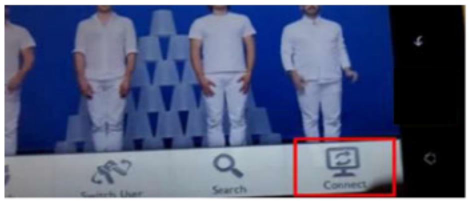
156. In my opinion, the YouTube Remote System discloses these limitations. To the extent the limitations are not disclosed, they are at least obvious in view of the general knowledge of a POSITA and/or the YouTube Remote Patent (*i.e.*, the '998 patent).

157. **Limitation 13.2:** In order to pair with one another a mobile device running the YTR application and a Screen both had to be connected to the Internet—which could be done by connecting to a user's home network (a "local area network") through Wi-Fi. *See* **Video #1** at 0:55-1:05 (stating that television and phone are connected to Wi-Fi). Indeed, at the time of the alleged invention of the '615 patent, smartphones, tablets, and televisions could connect to the Internet through a Wi-Fi connection. In order to connect to Wi-Fi these devices included a wireless network adapter, which would be a "network interface."

158. After the YTR application is connected to the local area network, the YTR application identifies Screens that have been paired to the session, including Screens connected to the Wi-Fi network, by receiving and processing a "loungeScreenConnected" message. In particular, each time a Screen on the local area network registers with the YT Lounge Server, the YT Lounge Server sends the YTR application a "loungeScreenConnected" message. *See supra*,

¶¶132-134. [redacted][35] The YTR application may be paired with multiple Screens and each time a Screen was paired to the session a "loungeScreenConnected" message was sent to the YTR application. (*supra*, ¶133). Thus, in my opinion the YTR application was able to identify playback devices that are connected to a local area network by receiving "loungeScreenConnected" messages.

159. **Limitation 13.3:** The YTR application displays on the graphical interface of the mobile device a selectable option for transferring playback from the control device, specifically the "Connect" button. In particular, to transfer playback to a Screen using the YTR application the user may press the menu button (shown in green in the above image) which brings up a "Connect" button (red) that the user may further select to transfer playback. **Video #1** at 1:04 to 1:07 (to transfer playback "you press menu then connect").



160. **Limitation 13.4:** The mobile device running the YTR application detects that the menu and Connect buttons have been pressed, and thus performs "detecting a set of inputs to transfer playback from the control device [the YTR application] to a particular playback device [one or more Screens that have been paired with the session]." **Video #1** at 1:04 to 1:07.

161. In my opinion, the Connect button being pushed constitutes a selection of the selectable option for transferring playback from the control device. For example, as I previously explained pushing the Connect button transfers playback from the mobile phone (*i.e.*, the control device) to the paired Screens. *See supra*, ¶¶136-141. Indeed, after a user has pressed the Connect button the YTR application displays a notice informing the user that it is "Connected to Leanback screen" and can control playback on the YTR application. **Video #1** at 1:13.

[35] In the July 12, 2011 version of the code that immediately precedes Sonos's alleged invention date, [redacted]

162. In my opinion the Connect button is also a selection of the particular playback device from the identified playback devices connected to the local area network. The YTR application identifies playback devices connected to the local area network by receiving a loungeScreenConnected message informing it that a Screen has joined the session. *See supra*, ¶¶157-158. Pressing the Connect button transfers playback to all the Screens that have been paired with the YTR application in a session, which would include the particular playback device from the identified playback devices (*e.g.,* one of the paired Screens).

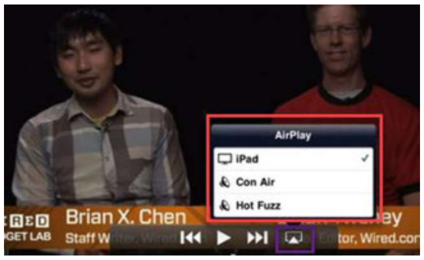
163. I have reviewed Sonos's Validity Contentions as it relates to the YouTube Remote and understand that for these limitations Sonos disputes only the YT Remote's disclosure of (1) "identifying playback devices connected to the local area network," and its disclosure of "(i) a selection of the selectable option for transferring playback from the control device and (ii) a selection of the particular playback device from the identified playback devices connected to the local area network." Validity Contentions at 184. Sonos's Validity Contentions provide no basis or explanation for why it contends this limitation is not disclosed by the YT Remote. To the extent Sonos is arguing that the YTR application must be capable of identifying each specific playback device that is connected and then display them on the screen so that they can be selected individually, I disagree with Sonos's interpretation of the claim for the reasons just mentioned.

164. Nevertheless, even under Sonos's interpretation this limitation would be obvious in view of the general knowledge of a POSITA and/or the '998 patent.

165. A POSITA would have understood that an application on a mobile device for controlling transfer, such as the YTR application, would benefit from being able to (1) identify playback devices available for transfer on the local area network and (2) display them on the user interface of the mobile device for selection by the user. For instance, a user may be in a particular room in their house and may want to transmit playback to only the Screen in that room, rather than paired Screens in other rooms. Thus, a POSITA would have been motivated to add to the YTR application the ability to display individual playback devices connected to the local area network so that a user could select a subset of devices for playback.

166. As I showed in Section IV, by July 15, 2011 (Sonos's alleged invention date), this feature was well-known technique and numerous prior art devices, including systems from Google (*e.g.*, Project Tungsten, *supra*, ¶¶27-28), from third-parties (Apple Airplay, *supra*, ¶30-32), and from Sonos itself (Sonos Controller, *supra*, ¶35) supported the ability to identify playback devices and individually select them for transfer. The functionality was also described in patent publications, for example the Al-Shaykh patent (*supra*, ¶33) and the YTR patent (*supra*, ¶29).

167. As just one example, on the right is an image showing an iPad with Apple's Airplay functionality. In the image, the Airplay icon (purple box) has been selected by a user which causes the iPad to show playback devices (red box) on the local area network (*e.g.*, "Con Air" and "Hot Fuzz") and allows a user to select one of those devices to transfer playback to. Ex. 12 at GOOG-SONOS-WDTX-INV-00000569. Additional discussion of the other devices is provided above in Section IV.



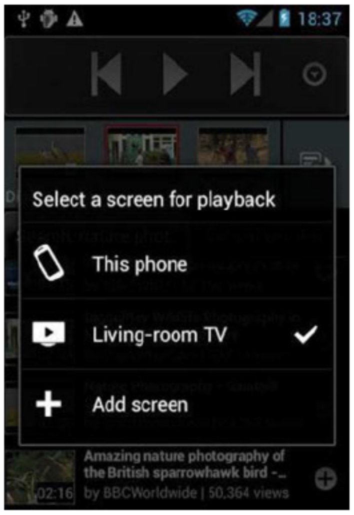
168. Additionally, this limitation was also described in the YouTube Remote Patent (the '998 patent). For example, the '998 patent discloses a "remote control" device (which may be an application running on a mobile phone) that may control a "controlled device" (such as a television). '998 patent at 5:42-45, 5:1-8. Remote controls and controlled devices are assigned "unique identifiers" in the system and a remote control may be paired with multiple controlled devices. 4:4-58, 8:1-10 ("a single remote control of the remote control 62 may be paired with any number of the controlled devices 64."). "[U]pon connecting to a network service, the remote controls and controlled devices may notify the network service that the remote controls and controlled devices are connected to the network," and the server may maintain a "session that includes the identifiers for all of the components sending and receiving messages (e.g., remote control(s) 14 and controlled device(s) 18)." '998 Patent at 4:21-57, 17:44-57.

169. The '998 patent expressly teaches that in some embodiments the "user interface" of the remote control may display the "previously paired controlled devices" in the session so that a user may select and control "one or more paired controlled devices":

> In some examples, the **user may also utilize the remote control application of remote control 75 to select one or more previously paired controlled devices**, and to send control messages to one or more paired controlled devices. **For example, the user may interact with user interface 84 and/or display 88 to interact with and control any available controlled devices**.

’998 patent at 10:62-11:6. In other words, the ’998 patent discloses that the remote control may identify the playback devices that have been paired to the session and then display them on its graphical interface so that the user may select one or all of them. *Id*.

170. It would have been obvious to modify the YouTube Remote application in view of the general knowledge of a POSITA and/or the teachings of the ’998 patent, such that the YouTube Remote System would identify each of the one or more Leanback Screens that have been paired to the session, and display them on its user interface so that a user may select one, all, or a subset of them. The combination would allow the YT Remote application to display available playback devices for selection, similar to what is shown in the Airplay image above (¶167). In fact, I have reviewed what I understand is a December 1, 2011 capture of the source code for the YouTube Remote. Levai Declaration, ¶7. The ability to select individual screens exists in this source code. Indeed, the December 1, 2011 capture of the source code for the YouTube Remote that I reviewed in this case includes the file [redacted] I also understand that the functionality was released in a January 2012 update, as can be seen in the image on the right, taken from a February 2, 2012 capture from the Internet Archive’s Way Back Machine. Ex. 14 (Way Back capture).


18:37
Select a screen for playback
This phone
Living-room TV
Add screen
Amazing nature photography of the British sparrowhawk bird -...
by BBCWorldwide | 50,364 views
02:16

171. A POSITA would have had a motivation to make the proposed modification and a reasonable expectation of success in implementing the combined system.

172. First, the ’998 patent is based on Google’s work on the YouTube Remote. It names

as inventors individuals that developed the YouTube Remote, and a POSITA would recognize that the ’998 patent was describing aspects of the YouTube Remote. The “remote control” and “controlled devices” in the ’998 patent may be a mobile device running the YTR application and a Leanback Screen (respectively), and the servers described in the patent may be Lounge Servers. The teachings of the ’998 patent are particularly applicable to the YTR system.

173. Second, modifying the YT Remote system based on the general knowledge of a POSITA and/or the YT Remote’s teaching would have simply used a known technique to improve similar devices in the same way. As just explained, by the time of the alleged invention of the ’615 patent one known technique for permitting a user to select a subset of available devices was to identify individual playback devices (*e.g.*, over a local area network), and then display an option on the graphical interface to allow a user to select one of those devices. Indeed, this was a known technique that was used in, for instance, Google’s Project Tungsten, Apple’s Airplay, Sonos’s prior art controllers and the Al-Shaykh patent publication. It is also disclosed in the ’998 patent. *See* Section IV.

174. Indeed, the YTR application already included the ability to control multiple Screens, and a POSITA would have understood that the device could be improved by allowing a user to select individual devices for playback. The proposed modification to the YouTube Remote would have been particularly straightforward because the YouTube Remote system already maintains unique identifiers for each Leanback Screen that has been paired with the YTR application in a session on the Lounge Server. Thus, in one exemplary implementation of the combined system the YTR application could receive the identifier that corresponds to each of the paired Screens from the Lounge Server, and then display them for selection on the user interface. The required changes would have been routine and required only software modifications.

(iv) Limitation 13.3: “after detecting the set of inputs to transfer playback from the control device to the particular playback device, causing playback to be transferred from the control device to the particular playback device, wherein transferring playback from the control device to the particular playback device comprises:”

175. In my opinion, the YT Remote system discloses this limitation.

176. For example, once a user clicks the "Connect" button on the YTR application, the playback of the video is transferred to and resumes on the TV screen. *See supra*, ¶136.

177. I have reviewed Sonos's Validity Contentions as it relates to Claim 13 of the YouTube Remote and understand that Sonos does not dispute that the YT Remote discloses this limitation. *See* Validity Contentions at 184-185. Sonos disputes only whether the "detecting" step of Limitation 13(e) is met, which I discussed above.

(v) Limitation 13.4:

"(a) causing one or more first cloud servers to add multimedia content to a local playback queue on the particular playback device, wherein adding the multimedia content to the local playback queue comprises the one or more first cloud servers adding, to the local playback queue, one or more resource locators corresponding to respective locations of the multimedia content at one or more second cloud servers of a streaming content service;

(b) causing playback at the control device to be stopped; and

(c) modifying the one or more transport controls of the control interface to control playback by the playback device; and"

178. In my opinion, at least under Sonos' interpretation of this claim limitation for purposes of its infringement contentions, the YT Remote system discloses this limitation.

179. According to Sonos, a "resource locator" may be a videoID and "adding the multimedia content to the local playback queue" does not need to store the actual multimedia content itself. *See* Section VI.2(b)-(c). In particular, Sonos accuses the following functionality in the accused YouTube Applications of satisfying this limitation: 1) when a user transfers playback the MDx server sends the playback device a setPlaylist message containing a videoID for the video that should be played (which is stored in a variable), 2) playback on the YouTube application is stopped, and (3) the YouTube application now controls the playback device.

180. To the extent Sonos's infringement contentions are credited, the YouTube Remote prior art satisfies this limitation in the same way. In fact, as it relates to this limitation, the only material difference between the prior art YTR system and the YouTube application is that the prior art YTR system actually stores a "local playback queue on the particular playback device" (an

array that stores the ordered list of videoIDs in the playlist), while the accused YouTube system does not (Section VI). *Compare supra* Ex. 10 (YT Remote API) at 3 ("setPlaylist" includes "set of video ids of any length, separated by commas").

181. Indeed, as I discussed above (*see supra*, ¶¶135-145) in the prior art YTR system upon transferring playback the YTR application caused a Lounge Server ("the one or more first cloud servers") to send a setPlaylist message the paired Leanback Screens ("playback devices") with a list of videoIDs for the playlist ("multimedia content" and "resource locators" according to Sonos). The list of videoIDs in the setPlaylist message are stored locally on the Screen as an array (a "local playback queue on the particular playback device"). Thus, limitation 13.4(a) is satisfied by the YTR application, to the extent Sonos's infringement contentions are credited.

182. Further, processing the setPlaylist message causes the YTR application to hide the local video player (CloudServerMessageListener.java, line 651), stop local playback (line 658), and update the local graphical interface to show player controls (line 657). *See supra*, ¶139. In other words, pressing Connect to transfer the video also causes the YTR application to stop local playback and modify its player control ("control interface") to control playback by the playback device. Playback is thereafter controlled using transport controls provided by the YTR application, including transport controls for skipping to the next track, the previous track, pausing the video, forwarding or rewinding the video, and setting volume of the video. This process is shown in the YouTube videos I discussed above. *see* ¶136; **Video #1** at 1:05-2:00. Thus, limitations 13.4(b) and (c) are also satisfied by the YTR application.

(vi) Limitation 13.5: "causing the particular playback device to play back the multimedia content, wherein the particular playback device playing back the multimedia content comprises the particular playback device retrieving the multimedia content from one or more second cloud servers of a streaming content service and playing back the retrieved multimedia content."

183. In my opinion, the YT Remote system discloses this limitation.

184. Again, once the user clicks the Connect button on the YT Remote application the playback of the video is transferred to and resumes on the TV screen. ¶136.

184. Additionally, the prior art YouTube Remote system discloses "playing back the multimedia content comprises the particular playback device retrieving the multimedia content from one or more second cloud servers of a streaming content service and playing back the retrieved multimedia content." For example, in 2010 (just as now), YouTube was a provider of streaming Internet videos. The YT Remote application retrieved YouTube videos from content servers using playback URLs, as I explained in ¶141. In particular, I understand that the playback URLs retrieve the videos from "Bandaid" servers on YouTube's Content Delivery Network. *See* Bobohalma Decl., ¶3, Levai Decl., ¶3. The Bandaid servers are "one or second cloud servers of a streaming content service," at least under Sonos's interpretation. Indeed, Sonos contends that the accused YouTube system satisfies this limitation by retrieving YouTube videos from "one or more CDN or 'Bandaid' servers." 2-24-2022 Supplemental Infringement Contentions, Ex. A at 88.

VIII. SECONDARY CONSIDERATION OF NON-OBVIOUSNESS

185. I have not seen any "secondary considerations" that may be used to demonstrate non-obviousness, or that would overcome the strong showing of obviousness based set forth above.

186. I understand that in response to Google's Interrogatory No. 4, asking Sonos to provide its contentions regarding secondary considerations, Sonos alleged that there was evidence of industry praise. I understand that to show "praise by others" there must be a nexus between the industry praise and the patented technology. Sonos points only to statements about GPM and YouTube Music; these products do not embody the patents. Moreover, the comments Sonos points to do not praise the claimed approach and only discuss playback transfer at a high-level.

187. To the extent that Sonos is permitted to set forth additional theories with respect to infringement or invalidity, I will address those additional theories in a reply declaration. Also, in this Declaration, I have focused on certain exemplary reasons Google does not infringe, but I reserve the right to raise additional reasons in subsequent reports that I serve in this case or at trial.

I, Samrat Bhattacharjee, declare under penalty of perjury under the laws of the United States that the foregoing is true and correct.

DATED: April 14, 2022

[illegible]

Samrat Bhattacharjee